UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 18: Advanced input/output features, saving and serialisation.

mihasan@liverpool.ac.uk

Previous lecture

- Different drawing/graphics approaches in Qt were discussed.
- Painting on widgets was discussed.
- Handling graphics using the Graphics View Framework was discussed.
- Classes introduced: **`QPainter`**, **`QPen`**, **`QBrush`**, **`QRectF`**, **`QTimer`**, **`QGraphicsView`**, **`QGraphicsScene`**, **`QGraphicsItem`**, **`QTransform`**

This lecture

- What is covered in this lecture?

  Enabling advanced input/output by the user in Qt applications.

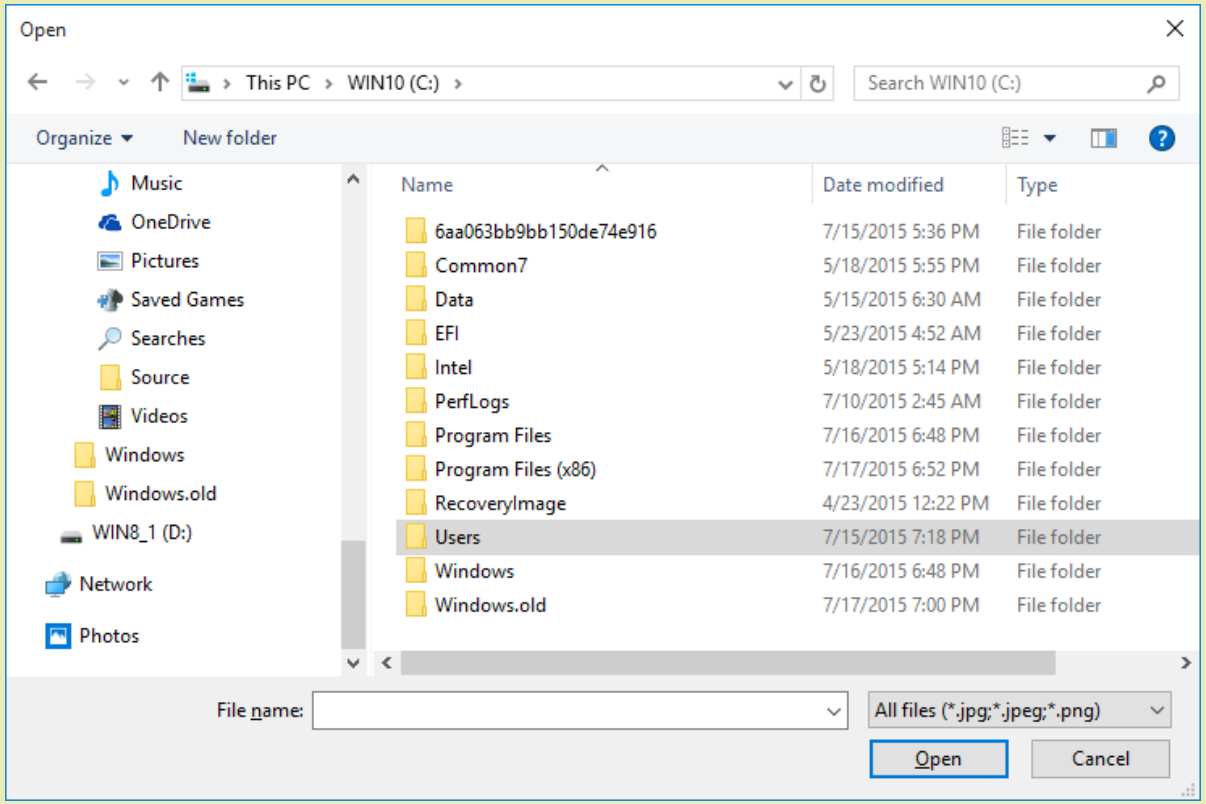
- Why it is covered?

  Loading/saving is a standard functionality in most applications nowadays.

- How are topics covered in this lecture: 2 Live demonstrations.

Custom input classes

- What if the user wants to have a “custom” input that is not a direct input?
- Qt has a range of input dialogs (all derived from **`QDialog`**):
- **`QFileDialog`**: Shows the window for opening a file of <u>any type</u>.
- It obtains a path to the required file the user inputs to the programme.
- Reference: https://doc.qt.io/qt-5/qfiledialog.html



Pointer to parent widget Title of the window

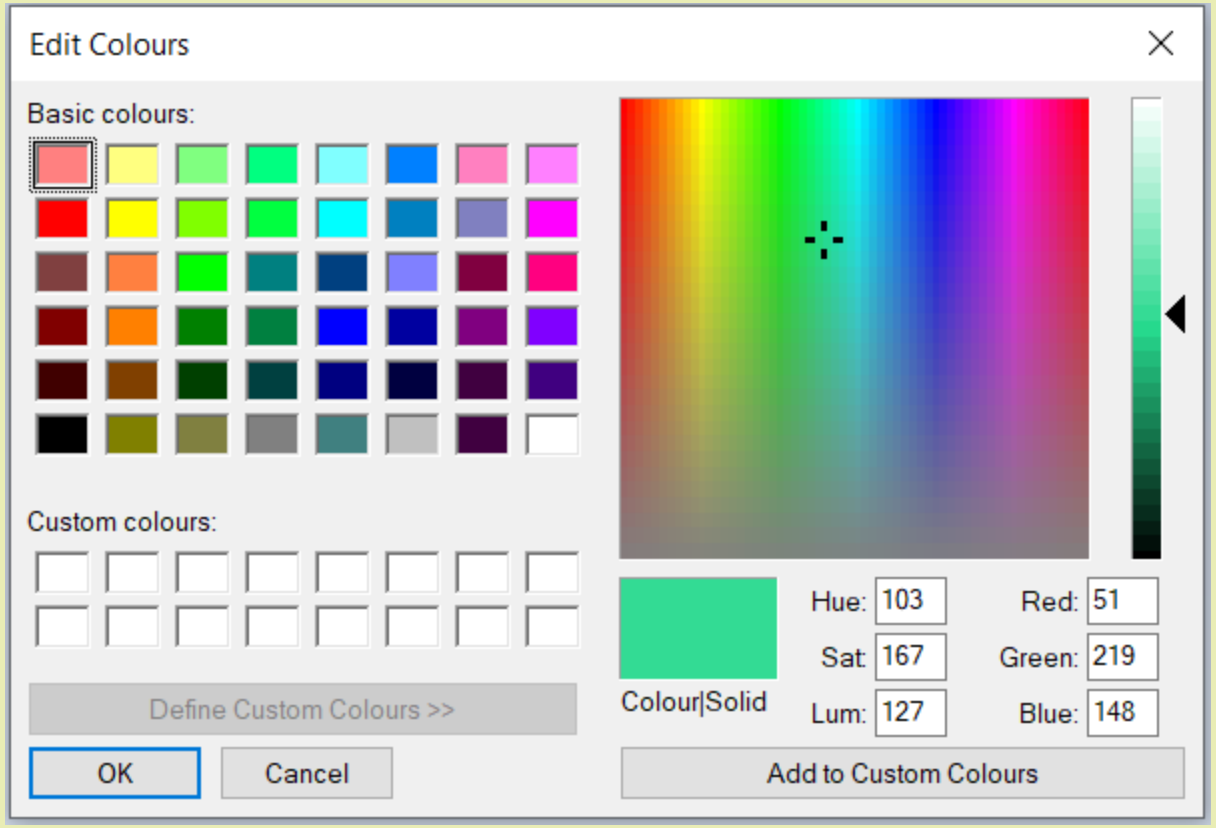
```
QString file = QFileDialog::getSaveFileName(this,tr("Save file"));
```

Custom input classes

- **`QColorDialog`**: Shows the window for selecting a colour.
- It returns an object of the class

  `QColor`.
- Reference:

https://doc.qt.io/qt-5/qcolordialog.html
- Example:




```
QColor user_clr = QColorDialog::getColor(Qt::white,this,tr("Get Color"));
```

Default color

Pointer to parent widget

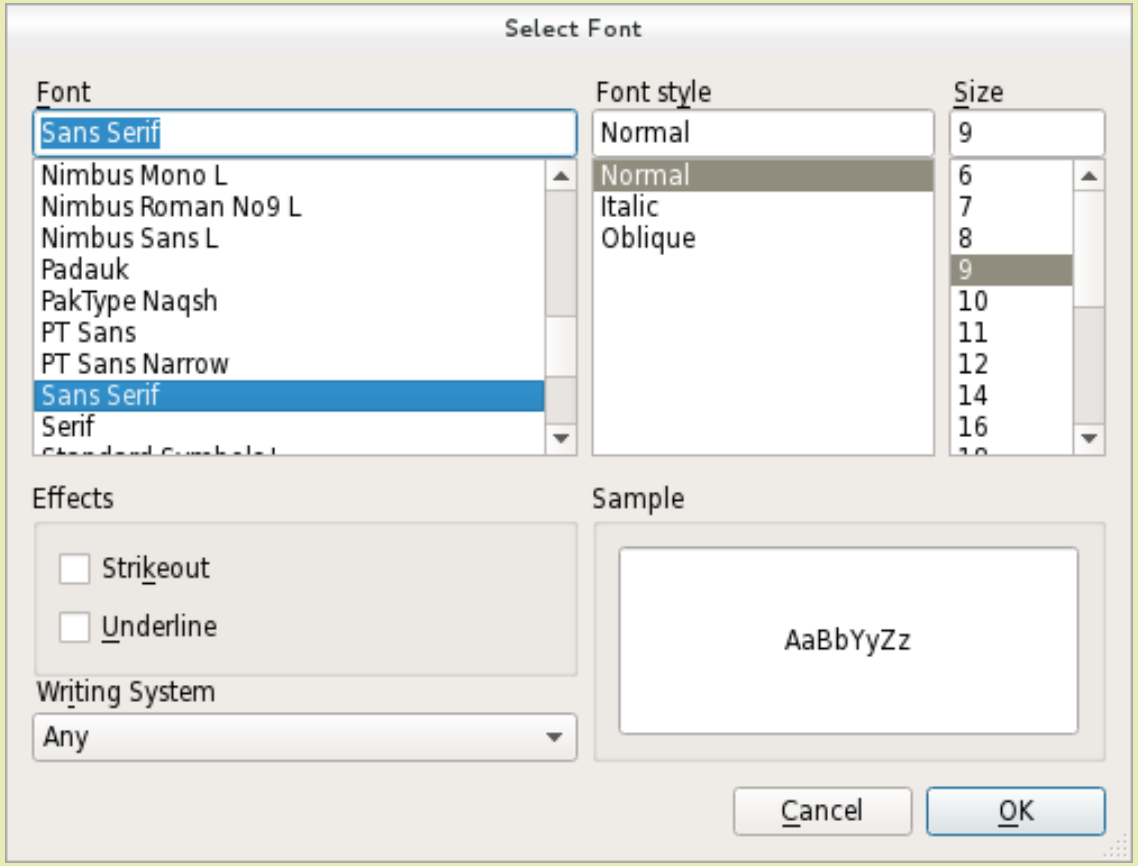
Title of the window

Custom input classes

- **`QFontDialog`**: Shows the window for selecting a font.
- It returns an object of the class

  `QFont.`
- Reference:

https://doc.qt.io/qt-5/qfontdialog.html
- Example:




```
QFont user_font = QFontDialog::getFont(&ok,QFont("Times",12), this);
```

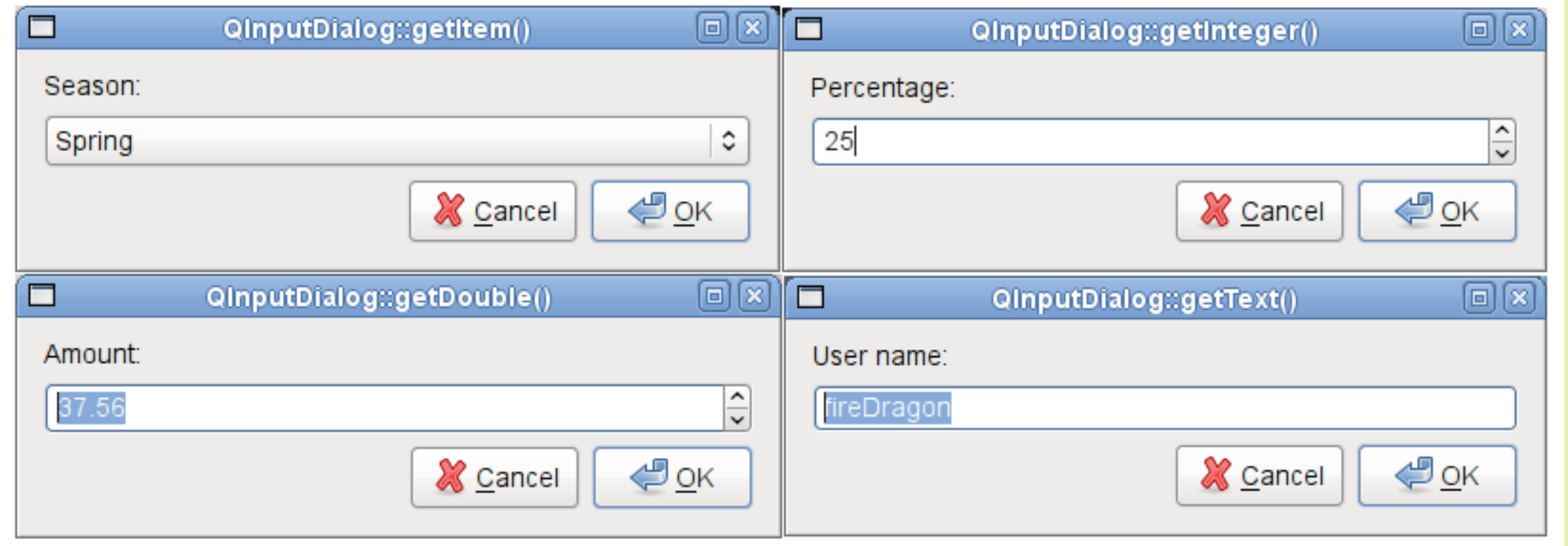
Reference to bool

Default font

Title of the window

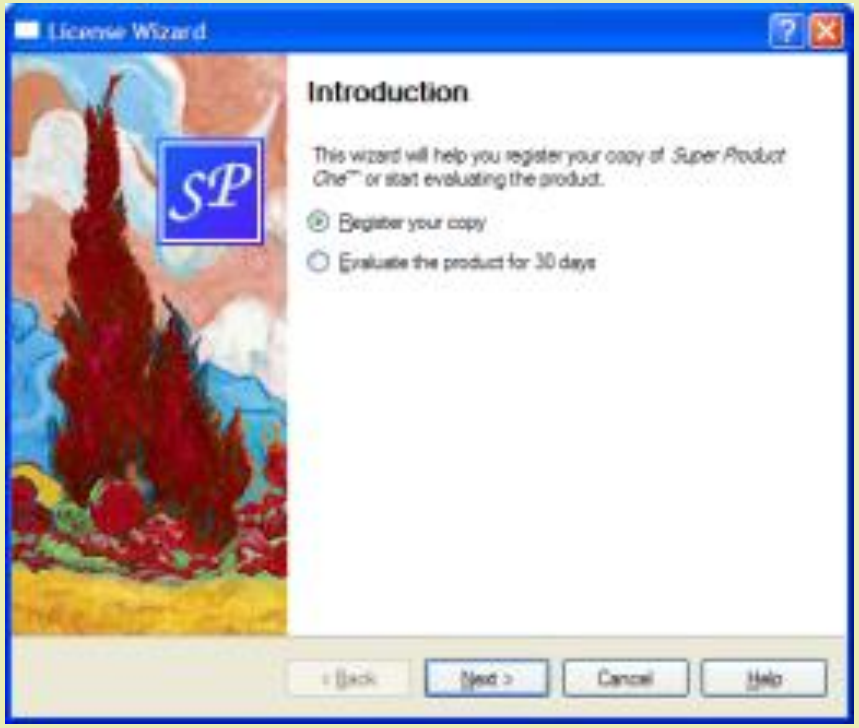
Custom input classes

- **`QInputDialog`**: Shows a simple pre-defined Dialog for simple input from the user
- It returns an object or a variable of any type:
- Reference: https://doc.qt.io/qt-5/qinputdialog.html



Custom input classes

- **`QWizard`**: Shows a pre-defined wizard interface.
- It offers different styles.
- Reference: https://doc.qt.io/qt-5/qwizard.html



Custom input/output classes

- **`QMessageBox`**: Provides a quick dialog to ask the user for an input or to display output to them
- Reference: https://doc.qt.io/qt-5/qmessagebox.html
- It is used to ask the user for more information, to display error and warning messages.
- The buttons in the message box, the text, and the signs/icons are all customisable.
- For error handling, Qt does not support the try-throw-catch mechanism (it has compatible with errors thrown by other libraries). Therefore a message box is the primary way to handle errors in Qt.

Good practice note: Use message boxes to handle errors when nothing can be done about them.

Custom input/output classes

- Some selected functions:

Member function	Is used to set:
setText	The message to display
setStandardButtons	The buttons available in the message
setWindowTitle	The title of the window of the message
setIcon	The icon in the message

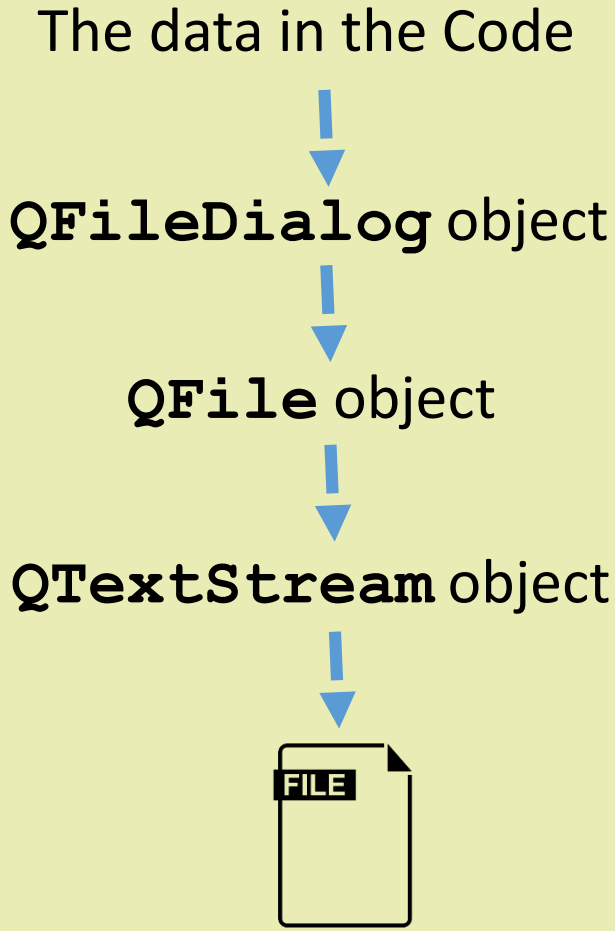
- Properties can be set when declaring the message box,

  or using the previous functions.

Loading and Saving text files

- The loading-saving functionality can be added to any application developed in Qt, using a combination of the following classes:

Class	Is used for
`QFileDialog`	Asking the user to specify the path and location of the file to be input/output
`QFile`	Creating a file/Opening a file
`QTextStream`	Reading and writing data to a file



`QFile` and `QTextStream` Classes

- The **`QFile`** class provides an interface for reading and writing files.
- Reference: https://doc.qt.io/qt-5/qfile.html
- An object of this class is the actual file that might be read or written.
- The **`QTextSream`** class provides an easy way to write, format and manipulate text streams.
- Reference: https://doc.qt.io/qt-5/qtextstream.html
- It can be used for file input and output, or it can be used for formatting local variables in the application.

A sample application: Notes saver

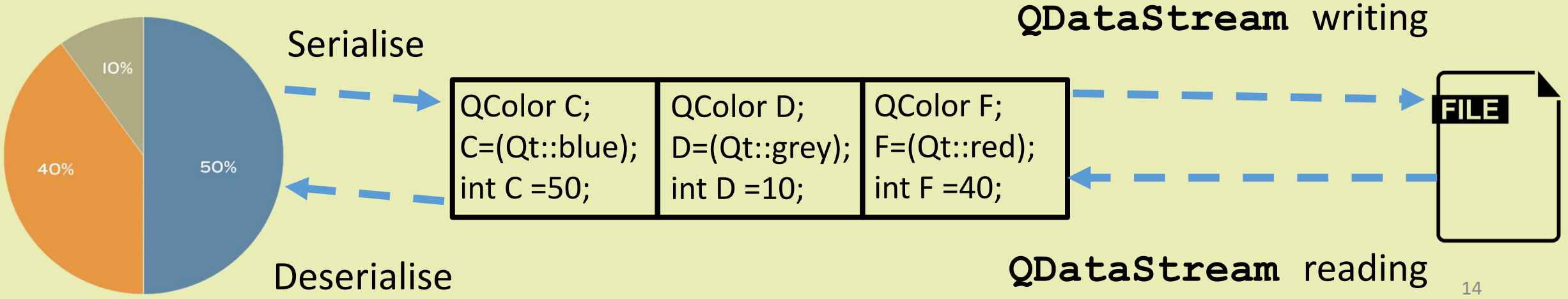
- The task: Design and implement a Qt-based GUI application that allows the user to save and load notes written as text to the application.
- Build the application (demonstration).

Good practice note:

- In Qt, it is best to define Loading/Saving as actions, with multiple ways of triggering these action (menus, toolbar, dialogs before closing the application), etc..
- Make sure the file is open before reading it, and make sure to close it after writing.

Serialisation

- How is it possible to save a drawing or a game?
- The vital step to save/load non-trivial file is to determine the information required to fully restore the application to the point where it is left.
- Serialisation is the process of converting objects into a one-dimensional stream of data to be saved. Then resort the original structure through “deserialisation”.



Serialisation

QUESTION: Identify the data that needs saving to restore the game to this status?

`QDataStream` Class

- This class provides an easy way to write, format and manipulate streams for any data type, or a combination of different datatypes.
- Reference: https://doc.qt.io/qt-5/qdatastream.html
- It serves an identical role of **`QTextStream`** but for other datatype.
- Make sure the order of extraction of data in deserlisation is the exact opposite of that in serialisation.

A sample application: Drawing saver

- The task: Draw few shapes using the graphics view framework and implement a serialisation function to save them. And a deserialisation function to load them.

- Build the application (demonstration).

Good practice note:

- Data containers such as **`QVector`** and **`QList`** are quite useful for saving non-text data.

Summary

- In this lecture, non-direct input/output of data to applications were discussed.
- For simple user inputs, many classes derived from **`QDialog`** are pre-defined to get user choices of colour, font, or files.
- Input / Output functionality of text files was discussed.
- The concept of serialisation was introduced, and input / output for arbitrary data types was discussed.
- Classes covered: **`QFileDialog`**, **`QFontDialog`**, **`QColorDialog`**, **`QInputDialog`**, **`QWizard`**, **`QTextStream`**, **`QFile`**, **`QDataStream`**