Assignment 3

1. Introduction

Please read this whole assignment tutorial letter before starting to ensure that you know what is expected of you and you do not get surprised by some requirement later in the development process.

Please note that you must not use Qt Designer, and you are expected to manually set up GUIs to ensure that you properly handle memory using Qt's parent-child functionality.

Use these to set up a project (using qmake, not cmake) that uses a `QMainWindow` as the user interface so that you can implement the functionality provided by a `QMainWindow` (menus, toolbar, status bar) to meet the requirements of the scenario given below.

Marks will also be awarded for following good programming practice, for example,

- naming conventions,
- code layout and line spacing,
- using initialiser lists in constructors,
- using forward class declarations in header files,
- enabling and disabling buttons as appropriate,
- GUI handling like setting focus, sequential tabbing, clearing input widgets (like text input fields being cleared and spin boxes being returned to some default value), and
- providing appropriate user feedback (including tooltips and messages to the user).

Your code should build and run without any warnings.

Consider the following scenario and then design a solution to meet the requirements listed.

2. Scenario

Transporting cargo around the world is essential in ensuring that customers have access to the goods they need and want.

Containers

All such items are packaged in some sort of container, each of which has a code, weight, and volume (which should be calculated and expressed as an integer). For the purposes of this scenario, there are two kinds of containers.

- A box, where we want to know its dimensions: length, breadth, and height.

- A cylinder, where we want to know its diameter and height.

The user should not be able to instantiate a simple container; only boxes and cylinders can be instantiated. Think carefully about how these requirements will be designed and an appropriate design pattern that could be used when instantiating container types. Use sensible ranges/limits for the input widgets on the user interface.

The container code, which is allocated when containers are created, takes the following format.

- The current year value between 2000 and 2099 (both included)
- Forward slash (/)
- The current month value between 01 and 12 (both included)
- Forward slash (/)
- A 'B' or a 'C', depending on the container type.
- A serial number starting from 1, running up to 9999. Each new container is given the next serial number.

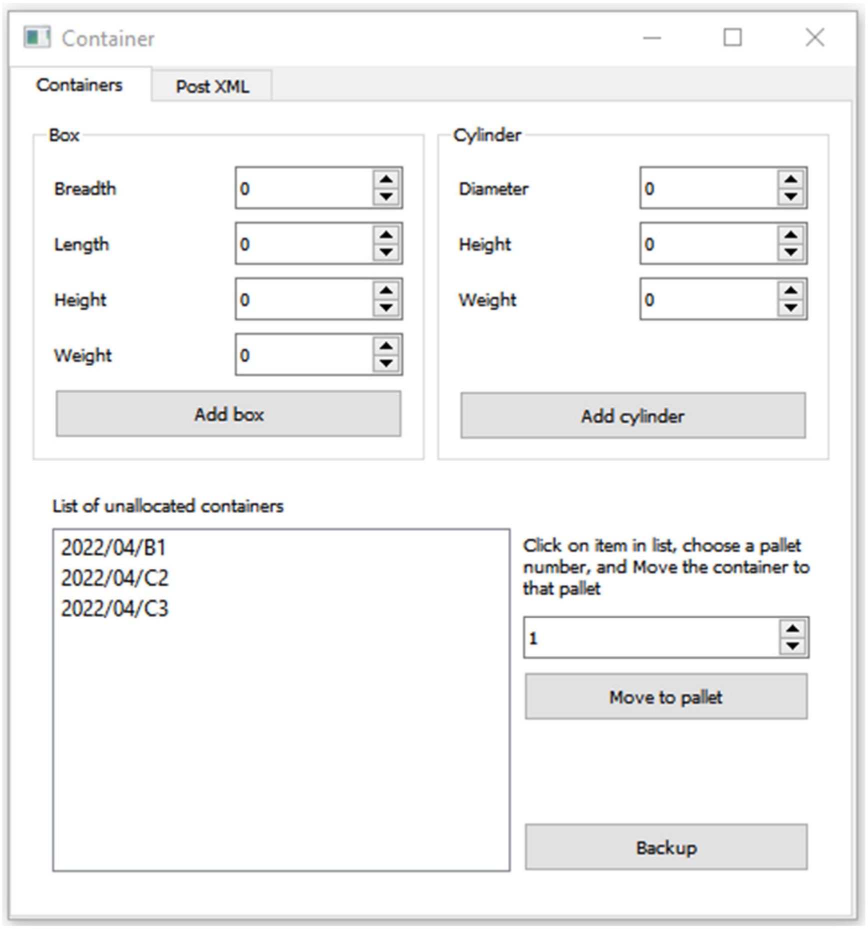
Examples of valid codes are `2022/01/B1` and `2022/01/C2`.

Lists

When a user creates a box or cylinder, the objects should be added to an unallocated container list. The container codes of these items should be displayed on the user interface using some model/view convenience class instance. The user should be able to backup this list of unallocated containers (remembering that you cannot simply back up the pointers to the container objects as they may be deleted before the restore is done); when restoring the unallocated container list, the container objects should be recreated and added to the list, and the model/view convenience object appropriately updated. Use an appropriate design pattern for this backup/restore functionality.

For transport, containers are packed onto a pallet (where it should be possible to calculate a pallet's total weight and volume); for the purposes of this assignment, you may assume that there is no limit to the number of pallets, number of items on a pallet, or a pallet's total weight and volume. The user should be able to click on a container code in the model/view convenience object, and this container should be moved to a pallet selected by the user, updating both the list of unallocated containers and the model/view convenience display. This means that there should also be a list of pallets, where pallets can be identified by a pallet number (which do not have to be sequential; a user can load containers onto pallets numbered 1 and 3 without loading any containers onto pallet 2).

Here is the required design of the user interface, showing how it could be used. As can be seen, all this functionality is available on one tab of the user interface.



List of unallocated containers

2022/04/B1
2022/04/C2
2022/04/C3

Click on item in list, choose a pallet number, and Move the container to that pallet

Serialising

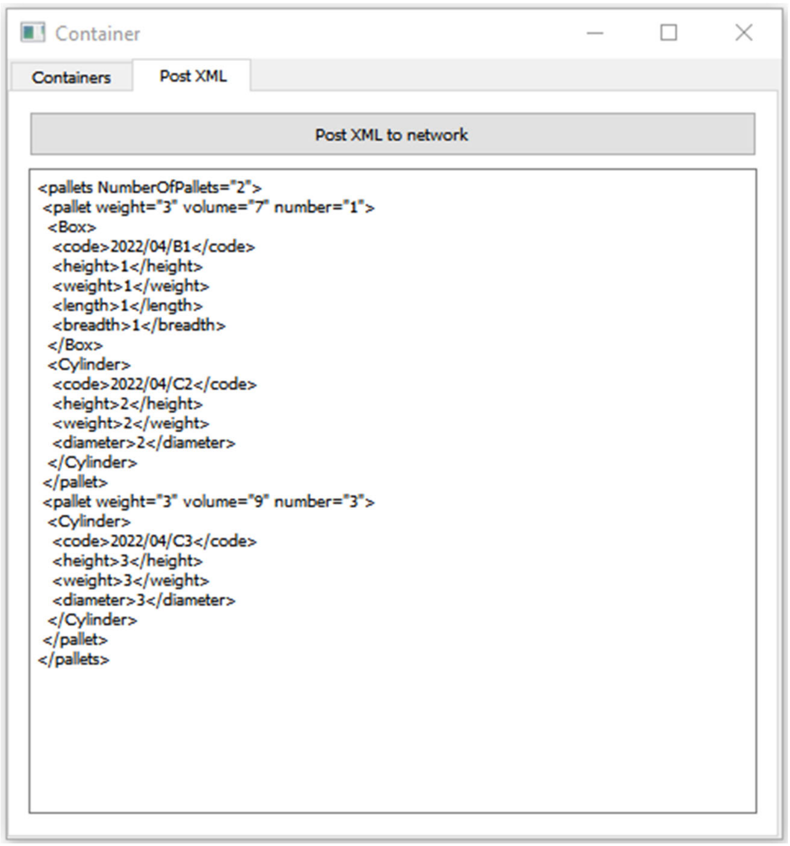
The user should be able to serialise the list of pallets. This is achieved on a second tab on the user interface.

- The serialising task should be run in a thread.
- The pallet list should be converted to XML for the serialisation task. Use container meta-objects to ensure a more generic approach to capturing container properties. See below for the required XML format.

```
<pallets NumberOfPallets="2">
 <pallet weight="3" volume="7" number="1">
  <Box>
   <code>2022/04/B1</code>
   <height>1</height>
   <weight>1</weight>
   <length>1</length>
   <breadth>1</breadth>
  </Box>
  <Cylinder>
   <code>2022/04/C2</code>
```

```
    <height>2</height>
    <weight>2</weight>
    <diameter>2</diameter>
   </Cylinder>
  </pallet>
  <pallet weight="3" volume="9" number="3">
   <Cylinder>
    <code>2022/04/C3</code>
    <height>3</height>
    <weight>3</weight>
    <diameter>3</diameter>
   </Cylinder>
  </pallet>
</pallets>
```

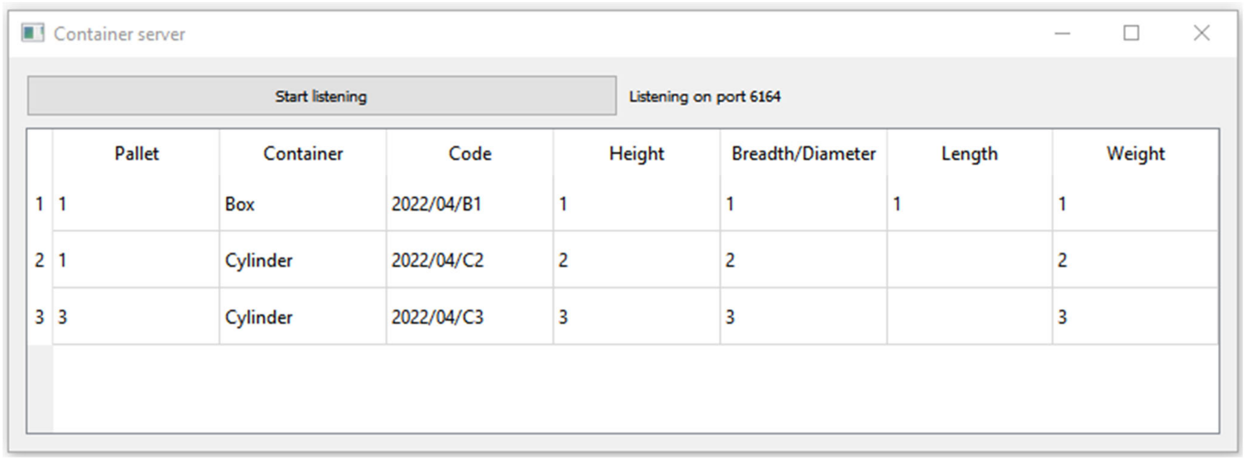
- This XML code is to be written to a TCP server over the network. Use 127.0.0.1/LocalHost for the host and 6164 as the port.
- Once the server has been successfully contacted, the XML code is to be written to an appropriate display widget on the tab. This indicates to the user that the serialisation is happening.
- Either use DOM to generate the XML code and QXmlStreamReader to parse it, or QXmlStreamWriter to generate the XML code and DOM to parse it.



```
<pallets NumberOfPallets="2">
 <pallet weight="3" volume="7" number="1">
  <Box>
   <code>2022/04/B1</code>
   <height>1</height>
   <weight>1</weight>
   <length>1</length>
   <breadth>1</breadth>
  </Box>
  <Cylinder>
   <code>2022/04/C2</code>
   <height>2</height>
   <weight>2</weight>
   <diameter>2</diameter>
  </Cylinder>
 </pallet>
 <pallet weight="3" volume="9" number="3">
  <Cylinder>
   <code>2022/04/C3</code>
   <height>3</height>
   <weight>3</weight>
   <diameter>3</diameter>
  </Cylinder>
 </pallet>
</pallets>
```

Container server

The server should be created as a separate project so that it can be run separately. The TCP server should listen for connections, and when an XML message is received, the XML code should be read, and the container data displayed using a model/view approach (not using a convenience class). Note that the server should check (using a regular expression) that the container code is valid, displaying **** if an invalid code is found. Also, when a new message is received, the display should be updated.



Listening on port 6164

	Pallet	Container	Code	Height	Breadth/Diameter	Length	Weight
1	1	Box	2022/04/B1	1	1	1	1
2	1	Cylinder	2022/04/C2	2	2		2
3	3	Cylinder	2022/04/C3	3	3		3

3. Requirements

The following general requirements should be noted.

- You should use menus and toolbars in your application.
- Pointers should be used for all instances of objects, and memory should be properly managed.
- Appropriate design patterns should be used where sensible.

4. Extras

Bonus marks will be added for using `QMainWindow` functionality.

- Splash screen
- Application icon
- About
- Help

5. Submission

Please check the Assignment 3 Notes page on myUnisa before submitting this assignment for information about how to submit the assignment.

- Use qmake, not cmake, for your assignment.
- Submit only the project folder. You should keep the `.pro` file in the folder, and delete the `.pro.user` file.

- You should not submit the build-desktop folder.
- Zip the two folders (for the two parts of the project) into one zipped file for submission.