UNIVERSITY EXAMINATIONS

UNISA university of south africa

October/November 2022

COS3711

Advanced Programming

80 Marks

Duration 2 Hours

EXAMINERS:
FIRST: DR CL PILKINGTON
SECOND: MR K HALLAND
EXTERNAL: DR L MARSHALL (UNIVERSITY OF PRETORIA)

This paper consists of 7 pages.

Instructions

1. You may type your answers in a word processor (and then print to PDF) or handwrite your answers (and then scan to PDF) for submission.
2. This is an open-book exam. Answer all questions. Please answer questions in order of appearance.
3. The mark for each question is given in brackets next to each question.
4. Note that no pre-processor directives are required unless specifically asked for.
5. Students must upload their answer scripts in a single PDF file (answer scripts must not be password protected or uploaded as “read only” files).
6. NO emailed scripts will be accepted.
7. Students are advised to preview submissions (answer scripts) to ensure legibility and that the correct answer script file has been uploaded.
8. Students are permitted to resubmit their answer scripts should their initial submission be unsatisfactory.
9. Incorrect file format and uncollated answer scripts will not be considered.
10. Incorrect answer scripts and/or submissions made on unofficial examinations platforms will not be marked and no opportunity will be granted for resubmission.
11. A mark awarded for an incomplete submission will be the student’s final mark. No opportunity for resubmission will be granted.
12. A mark awarded for illegible scanned submission will be the student’s final mark. No opportunity for resubmission will be granted.
13. Only the last file uploaded and submitted will be marked.

14. Submissions will only be accepted from registered student accounts.

15. Students who have not utilised invigilation or proctoring tools (IRIS) will be deemed to have transgressed Unisa's examination rules and will have their marks withheld.

16. Students must complete the online declaration of their work when submitting. Students suspected of dishonest conduct during the examinations will be subjected to disciplinary processes. Students may not communicate with other students or request assistance from other students during examinations. Plagiarism is a violation of academic integrity, and students who do plagiarise or copy verbatim from published work will be in violation of the Policy on Academic Integrity and the Student Disciplinary Code and may be referred to a disciplinary hearing. Unisa has zero tolerance for plagiarism and/or any other forms of academic dishonesty.

17. Students are provided 30 minutes to submit their answer scripts after the official examination time. Students who experience technical challenges should report to the SCSC on 080 000 1870 or their College exam support centres (refer to the Get help during the examinations by contacting the Student Communication Service Centre (unisa.ac.za)) within 30 minutes. Queries received after one hour of the official examination duration time will not be responded to. Submissions made after the official examination time will be rejected by the examination regulations and will not be marked.

18. Non-adherence to the processes for uploading examination responses will not qualify the student for any special concessions or future assessments.

19. Queries that are beyond Unisa's control include the following:
    a. Personal network or service provider issues
    b. Load shedding/limited space on personal computer
    c. Crashed computer
    d. Using work computers that block access to myExams site (work firewall challenges)
    e. Unlicensed software (eg license expires during exams)
    Postgraduate students experiencing the above challenges are advised to apply for an aegrotat and submit supporting evidence within ten days of the examination session. Students will not be able to apply for an aegrotat for a third examination opportunity. Postgraduate/Undergraduate students experiencing the above challenges in their second examination opportunity will have to reregister for the affected module.

20. Students experiencing technical challenges should contact the SCSC on 080 000 1870 or via e-mail at mailto:Examenquiries@unisa.ac.za or refer to the Get help during the examinations by contacting the Student Communication Service Centre (unisa.ac.za) for the list of additional contact numbers. Communication received from your myLife account will be considered.

Remember to complete the Honesty Declaration when submitting your answers. By submitting your answers you are confirming that your submission is your own, unaided work.

Most people like chocolate. “Chocolate is a food made from roasted and ground cacao seed kernels, that is available as a liquid, solid or paste, on its own or as a flavouring agent in other foods” (https://en.wikipedia.org/wiki/Chocolate). Also, as most dark chocolates are vegan, they can be enjoyed by people with stricter dietary choices. Chocolate can also be found in many forms: bars, slabs, and in drinks, to name just a few.

For the sake of this scenario, you may assume the following about designing an application that manages various types of chocolate.

- Initially it has been decided that chocolates can come in bars and slabs. A user should not be able to create instances of a basic chocolate that is neither a bar nor a slab.
- All chocolates have a name (in the form of a string) and a flag indicating whether it is vegan or not. They should also all have a string representation of their data members.
- You are required to include the number of blocks there are for a slab chocolate.
- Each instantiated chocolate should have a unique serial number, starting at 1 and incrementing by 1 for each new chocolate instantiated. This should not be managed outside the chocolate class hierarchy.
- There is a register of chocolates that holds all instantiated chocolate objects.
- An appropriate design pattern should be used that allows the user to choose the kind of chocolate to instantiate.

Question 1 **[29 marks]**

1.1 Considering the scenario given above, draw a partial UML class diagram that captures the scenario. You should include the necessary classes, class attributes, and class relationships that are mentioned in the scenario, including class member access specifiers (+, -, #). You do not have to include the Client/GUI class nor indicate constructors or other methods in the classes you specify.
[You may use a software tool to create the UML class diagram. Use underlining to represent italics in hand-drawn UML class diagrams.] (14)

1.2 Write the class declaration and implementation code for the `Chocolate` class. Include the following requirements when doing so:

- All requirements from the scenario on page 2: instantiation restrictions, string representation (that returns the data values in a comma-separated list), and unique serial number.
- There should be only one constructor that will allow for optional arguments for the data members. (10)

1.3 Based on the scenario, is `Chocolate c;` valid code? Explain clearly why you say so. The mark is only awarded for the reasoning. (2)

1.4 It has been decided to make the `Chocolate` class a `QObject`.

```
class Chocolate : public QObject
{
   …
};
```

1.4.1 Give one major benefit of making `Chocolate` inherit from `QObject`. (1)

1.4.2 Besides what is given in the scenario before question 1, assume that `CouvertureChocolate` inherits from `Chocolate`. Would the following code (that passes an instance of `CouvertureChocolate` to a `display()` function) be legal? Explain clearly why you say so. The marks are only awarded for the reasoning.

```
CouvertureChocolate cChoc;
display(cChoc);
```

(2)

Question 2 **[22 marks]**

The intention is to scan text files for references to vegan chocolates. This task will be taken on by a class that is to be run as a thread.

2.1 Write the class definition of the `Scan` class, which is to be run as a thread, that will be used to count the number of times vegan chocolates is mentioned in a piece of text. The filename to scan will be passed to the class instance, and the number of occurrences should be returned to the calling function once the scan is complete. Include all functionality that would be required so that it conforms to best practice. Use function and variable names that will make it clear what the purpose of the function or variable is. (7)

2.2 It has been decided to use regular expressions to scan the file. Write the regular expression, in quotes, that can be used to check for the words “vegan chocolate” or “vegan chocolates” in the provided text file. You should not use the pipe symbol (|) and ensure that you use escape characters correctly. (4)

2.3 The following code stub is used to scan a list of files and collect the data returned from the scanning process.

```
QStringList listOfFiles; // contains list of all files to be scanned
foreach (QString filename, listOfFiles)
{
    // use threading to scan the file
    // collect the number of mentions in a QList<int> named "counts"
}
```

This data is to be written to an XML file in the following format.

```
<fileCounts>
   <file name="filename_1">          the name of the first file in the list
      <count>23</count>              the count for this file
   <file name="filename_2">          the name of the second file in the list
      <count>5</count>               the count for this file
   . . .                             and so on
</fileCounts>
```

Using the partial code given below, and assuming that `QStringList listOfFiles` and `QList<int> counts` are accessible by this code, complete the code as indicated so that the required XML text above will be generated using DOM.

```
/*type*/ domDoc;
QDomElement root = /*function*/("fileCounts");

// set up any initial variables if required

// loop through all the items adding them to the XML text as necessary
```

(11)

Question 3 **[14 marks]**

Ignoring bars and slabs of chocolate, the chocolate manufacturer wants to have a range of dark, milk, and white chocolate as sub-types of chocolate.

The following constructors are to be used.

```
DarkChocolate("NightTime", "yes", 85);  // passing chocolate name, vegan status, and
                                        percent chocolate
MilkChocolate("Jersey", "no");          // passing only chocolate name and
                                        vegan status
WhiteChocolate("Fairland", "no", "no"); // passing chocolate name, vegan status, and
                                        whether it contains vanilla
```

It is required that the following be implementable (one function to create these three types of chocolate).

```
Chocolate *choc = createChocolate(/* pass the type, name, vegan status and
optional third parameter*/);
```

3.1 Which design pattern is being suggested here? (1)

3.2 Write the class definition for the class that would be used to implement this design pattern based on the scenario at the start of this question. (4)

3.3 The following is a class that that maintains a list of types of `Chocolate`s created by the `createChocolate()` function.

```
class ChocolateList
{
public:
    ChocolateList();
    void addChocolate(/*type*/);
private:
    QMap</*key*/, /*value*/> darkChocolates;
    QMap</*key*/, /*value*/> milkChocolates;
    QMap</*key*/, /*value*/> whiteChocolates;
}
```

Considering the scenario at the start of this paper (on page 3), answer the following questions.

3.3.1 What parameter would be passed to the `addChocolate()` function? (1)

3.3.2 In the `ChocolateList` class, there is only one `addChocolate()` function with one parameter, but three lists, one for each kind of chocolate. Without adding any further data members to any of the classes in the `Chocolate` hierarchy but allowing the `Chocolate` class to inherit from `QObject` (as in question 1.4), how would you propose the `addChocolate()` function determine which of the three lists a `Chocolate` instance passed to the class should be added? (2)

3.3.3 What piece of data would you use as the key for the `darkChocolates QMap`, and why do you say so? (2)

3.3.4 It has been decided that a user should only be able to create one instance of the `ChocolateList` class. Rewrite this class so that this requirement is met using a standard design pattern. (4)

Question 4 **[15 marks]**

A model-view approach can be used to manage and display a list of instantiated `Chocolate` object data.

4.1 Discuss three different general techniques that are open to a Qt programmer to manage and display this data using a model-view approach. Explain how the technique would be implemented (mentioning specific Qt classes as examples that may be appropriate in this scenario), including what other factors need to be considered when implementing the technique. (9)

4.2 Suppose there is a `GUI` class (which is the main user interface), and a `MyModelView` class that implements the model-view approach proposed in this question. If you were to use the Memento pattern to set up a backup and restore facility for `MyModelView`, answer the following questions.

4.2.1 Which class would create the memento? (1)

4.2.2 Which class would contain the data that is to be backed up? (1)

4.2.3 Which class would hold on to the created memento? (1)

4.3 Given the following declarations

```
class Window : public QMainWindow
QTcpServer *tcpServer
QTcpSocket *tcpSocket
```

reorder the following statements into the order in which they would be performed by a TCP server application listening for, and processing, connections. Remember that `connect()` statements may have limited/function scope, so you may assume that you cannot list them all at the start of the application. You may simply write down the numbers of the statements in the correct order.

```
1. connect(tcpSocket, &QTcpSocket::readyRead, this,
                                                &Window::readMessage);
2. connect(listenForConnection, &QPushButton::clicked, this,
                                                    &Window::setUp);
3. connect(tcpServer, &QTcpServer::newConnection, this,
                                            &Window::handleConnection);
4. QByteArray data(tcpSocket->readAll());
5. tcpServer->listen(QHostAddress::Any, 5555)
6. tcpSocket = tcpServer->nextPendingConnection();
```
(3)