Faculty of Technology – Course work Specification 2020/21

Module name:	Functional Software Development		
Module code:	CTEC3904		
Title of the Assignment:	Practical Assignment		
This coursework item is: (delete as appropriate)	Summative		
This summative coursework will be marked anonymously		No	

The learning outcomes that are assessed by this coursework are:

1 Analyse and critically review the principal concepts of functional software design
2 Critically evaluate the support for the application of functional software design in the context of a contemporary programming language
3 Apply functional software design to produce a software solution to a practical problem
4 Analyse a given functional software solution in terms of relevant performance criteria

This coursework is: (delete as appropriate)	Individual	

This coursework constitutes 100% **to the overall module mark.**

Date Set:	Monday 1st February 2021
Date & Time Due:	Monday 19th April 2021 no later than 12 (noon)

Your marked coursework and feedback will be available to you on: If for any reason this is not forthcoming by the due date your module leader will let you know why and when it can be expected. The Head of Studies (headofstudies-tec@dmu.ac.uk) should be informed of any issues relating to the return of marked coursework and feedback. Note that you should normally receive feedback on your coursework by **no later than four working weeks after the formal hand-in date,** provided that you met the submission deadline.	18th May 2021

How to submit your work. *Please note:*

- *There are two things to upload – please follow the instructions below carefully.*
- *Your submission is not complete without both of these uploads.*

1. **A report to be submitted to Turnitin.**

   Your solution to both tasks (see below) should be written up and presented within a single Word document (or pdf derived from some other text markup such as LaTeX for example). Take care to present your work clearly and concisely. There is not a specific page limit because we want you to include diagrams and these can vary considerably. However, as a *guideline* we would expect each task to be presented over two-three pages. The second task may be longer if the code outputs are more extensive in which case you can put an Appendix section at the back of the document and reference the output listings within the main body of the text. The use of diagrams will be especially helpful to demonstrate your understanding so feel free to use up extra pages if you need to draw lots of pictures – but **be careful not to dump lots of irrelevant material**. It is a balance that needs to be struck. Ask yourself if you have described your solution fully, but not cluttered it up with repetitive or extraneous material. The curation of the material and its presentation is part of the assessment: provide what you need but no more.

2. **On Blackboard upload your Scala code:** Document.scala.

   Only upload the one file. We will not use this to mark the assessment directly because you will have included the main parts for discussion into your report. However, we want the original file in case we have any subsequent queries about your solution or in case we do want to run it to check something out. Therefore, handing in the source code provides us with the code upon which your report is based. Please make sure that it is consistent with version of the code you describe in your report.

Late submission of coursework policy: Late submissions will be processed in accordance with current University regulations which state:
*"the time period during which a student may submit a piece of work late without authorisation and have the work capped at 40% [50% at PG level] if passed is **14 calendar days**. Work submitted unauthorised more than 14 calendar days after the original submission date will receive a mark of 0%. These regulations apply to a student's first attempt at coursework. Work submitted late without authorisation which constitutes reassessment of a previously failed piece of coursework will always receive a mark of 0%."*

Academic Offences and Bad Academic Practices:

These include plagiarism, cheating, collusion, copying work and reuse of your own work, poor referencing or the passing off of somebody else's ideas as your own. If you are in any doubt about what constitutes an academic offence or bad academic practice you must check with your tutor. Further information and details of how DSU can support you, if needed, is available at:

http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/academic-offences.aspx and http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/bad-academic-practice.aspx

Tasks to be undertaken: See (following) attached document.

Deliverables to be submitted for assessment: See (following) attached document.

How the work will be marked: See (following) attached document.

Module leader/tutor name:	**David Smallwood**
Contact details:	**drs@dmu.ac.uk (GH6.70)**

Important note

Please do **NOT** be tempted to search the internet for an existing solution and then hand this in – e.g. if you run out of time. This is cheating – it is better to hand in what you have honestly achieved by yourself than try to get credit for someone else's work. Cheating is an **academic offence**.

If you get stuck then please ask the module tutors for assistance.

Do **NOT** use anyone else's material without referencing it – this is **bad academic practice** or **plagiarism**. These are considered as **academic offences**.

Do **NOT** work jointly on a solution; do **NOT** give your solution to anyone else to "help them" and do **NOT** accept anyone else's solution as "guidance". Such practice could lead to an allegation of **collusion** which is an **academic offence**. **All parties** involved in collusion (the givers, the receivers, the collaborators) can be found guilty of an academic offence, irrespective of motive.

Requirements

1. Download and install

Download from Blackboard and install the Scala object Document.scala. It expects to be installed in a package called coursework:

src->coursework->Document.scala	(using Eclipse Scala/IDE)
src->main->scala->coursework->Document.scala	(using IntelliJ)

In the same package place the associated file DocumentData.scala. NB This file only contains some sample data to use within the program but it has been placed in a separate file to avoid cluttering up the code inside Document.scala.

2 Check it runs

Once you have installed the program you should run it. Have a look at main and see the experiments that are available – you will need to uncomment them individually to avoid getting too much output all at once. Just get a feel for the kind of application this is – look at the code and look at the output.

3 What is it?

The file consists of two data structures (classes); one called **Section** and another called **Book**. The idea is that a book is made up of a series of chapters (i.e. main sections) and that each of these sections can be split into subsections, etc.

```
Book: Title
        Chapter 1
                Subsection 1.1
                        Subsubsection 1.1.1
                        Subsubsection 1.1.2
                        …
                Subsection 1.2
                        Subsubsection 1.2.1
                        Subsubsection 1.2.2
                        …
                …
        Chapter 2
                …
        Chapter 3
                …
        Etc.
```

The Book part is simply a list of Sections (Chapter 1, Chapter 2, …). Each section is implemented as a tree with a root node and any number (i.e. a list) of subsections. If a section has an empty list of subsections then it is a leaf in the tree. We are referring to "Chapter", "Section", "Subsection", and so on in this discussion because they relate to how we think about organizing books. However, from the point of view of the code, each of these is simply an instance of class **Section**. Therefore, at a very practical level, this coursework is really about manipulating trees and lists of trees. All of the data structures are immutable and recursion is used to perform many of the operations. Higher order functions on lists (e.g. map and filter etc.) are also used to process the lists of trees.

The techniques here are applied to our simplified model of books. However, this is essentially an *n-ary tree* that could be used to encode, e.g., XML, JSON, LaTeX, or any other type of markup script, and there are lots of other examples of *n-ary trees* including, e.g., filesystems. So the code here that you study and that you write are examples of a style of coding that have much wider application.

When you run the code you will see that it is printing out documents with sections and subsections which are numbered. The documents themselves are specified within the **DocumentData** file so you could create some more Book or Section values of your own to try as well as the ones provided. However, we recommend you go with the data provided first of all until you are more familiar with the case study.

We recommend that you study the code before moving on to part 4 next.

4 What you need to do

We suggest you undertake the tasks below carefully in the order specified. It is likely that your understanding of the scenario will develop while following the steps in this order.

1. Task 1 (weight: 60%) *This task is designed to assess your understanding of a program written in the FP style. It requires you to demonstrate your understanding by explaining in detail a variety of coding constructs in the context of a specific example. We offer a choice of methods so that you can select the one you feel would enable you to express your understanding the best.*

   Experiment with the code and try the examples. Then read the implementation of class **Section**. It contains a number of methods, some of which are listed in the table below:

Method (Choose ONE of these)
toString
performSectionNumbering
insertSubsection
deleteSubsection
getSubsection

   Your first task is to choose ONE of these methods and explain it in detail. You should use diagrams to illustrate your answer. We want to see that you understand *every piece* of the code in your chosen method and that you can explain what its purpose is. For example, if there is an **if** then we want to know that you understand not only what the **if** condition says but also that you know *why* that condition is important with the particular values it has. To do this task well requires that you develop a good knowledge of the data structure. This is not something you can do superficially. You may need to research the syntax as well as the methods/operators/higher order functions used.

   Feel free to comment upon how you might improve the method or simply how you would have reformatted/restructured it if you feel this is a relevant point. We want you to be critical but, at the same time, to demonstrate that you understand the code you are explaining in detail, and in the context of the example.

Please avoid generic "text book" answers such as "a map function applies a function to each element of a list". Such a statement, unqualified, will not be worth very much. The main purpose of asking you to explain the methods within the context of the example is so that you can demonstrate an understanding of *why* they are appropriate in this context as well as *what* operation they perform.

How we will mark it

When assessing your answer to this part we will follow the marking scheme below. Please remember that at final-year level we are not just looking for "right" vs "wrong" answers. We are looking for the depth of insight and understanding revealed by your answer. You will need to present your work carefully in order to convey the depth of understanding and insight that you have. We cannot assume this depth of knowledge – we need to see evidence of it clearly shown in your answer.

The marking scheme allows us to identify the most relevant category (below) which, in turn, reflects the criteria outlined in the DMU mark descriptors (https://www.dmu.ac.uk/about-dmu/quality-management-and-policy/academic-quality/learning-teaching-assessment/mark-descriptors.aspx). The generic criteria have been adapted for this particular coursework. The marker will form an opinion as to where within the range of marks for a particular category the final mark should be.

Marking scheme

Does the answer overall represent a *reasonable* attempt to explain the method?

- Absolutely/yes/just about.
    - **No fault can be found** other than very minor (e.g. typographical) issues. Displays **exceptional** analytical skill and **exceptional** presentation skill. 90-100
    - An **outstanding** solution evidenced by **sound and insightful** critical analysis and **excellent** presentation. 80-89
    - An extremely **well-developed** solution showing an **authoritative grasp** of the underlying concepts. It must be **very well presented**. 70-79
    - A **detailed** solution demonstrating a **thorough grasp** of the underlying theory. Clear evidence of **insight** and **critical judgement** in selecting, ordering and anlaysing content. It must be **well presented**. 60-69
    - An **effective** solution demonstrating evidence of a **clear grasp** of the concepts. The solution must demonstrate an ability to construct and organize the relevant arguments. There must be **some degree** of critical analysis. It must be **clearly presented**. 50-59
    - The solution demonstrates understanding of **basic** concepts and perhaps not all of the relevant concepts. It has **limited** analytical content and may be rather **derivative** or **simplistic** in its arguments. The **presentation should be sufficient** to convey this basic knowledge and understanding. 40-49
- No/not really. *In the following, even a reasonable attempt at presentation, i.e. layout, choice and organization of material on the page, cannot compensate for a lack of basic knowledge and/or a lack of appropriate analysis.*
    - The solution represents overall an **insufficient** response. It addresses **some** elements but overall indicates **weakly-developed** arguments or **significant gaps**

in understanding. The presentation may be **weak**. 30-39

- A **poor** solution which falls **substantially short** of the expected level of analysis and presentation. It demonstrates **little knowledge** and/or **little evidence** of appropriate argument. 20-29
- A **very poor** solution demonstrating **very few** relevant facts and **only isolated** understanding. The solution most likely has **little adherence** to the task. 10-19
- An **insufficient** response. Displays virtually **no knowledge or understanding**. The work submitted may be **inappropriate** or **irrelevant** or **missing**. 0-9

2. Task 2 (weight: 40%) *This task is designed to assess your ability to write a method in context using the FP style. It is not sufficient just to write the method – we want you to provide evidence of the method working (show us some of your test data and output) and to discuss your solution – to defend it.*

   Implement ONE of the following methods within the **Book** class:

Method (Choose ONE of these)	**Difficulty**
shorten	low
replaceChars	low
swapChapters	medium
insertAtZero	high
Your own method	?

Choose ONE of the methods. You can make up your own method if you want to be inventive. But use this option carefully. We will decide the level of difficulty of the method you define and implement compared to those given above (our decision is final). You might choose this option for a couple of reasons: (i) you want to demonstrate your creativity by thinking of some interesting example of your own; or (ii) you would prefer to concentrate on a more straightforward method of your own choosing. The latter case may be an option if you are less confident about producing an answer to a higher-difficulty method and you would prefer to demonstrate something working while acknowledging that it won't attract the highest marks.

Demonstrate, analyse, and critically review. The assessment is not just about *writing* the code. We want to you to explain the thought process behind your solution. We want you to explain to us why you solved the problem in the way that you did. In particular, you should indicate how the intermediate stages of your solution work to transform the data structure towards the result – use of diagrams here would be especially helpful. In short, a piece of undocumented code with no explanation will not score very well at all. At this level we are looking for your analysis and understanding over and above just writing the code itself.

Present it well. We neither want nor require you to attach lots of separate files of output data. You should choose some example runs that show it working properly and include these within your report. To help you to demonstrate your work most effectively you should pay particular attention to the way in which the sample runs are presented on the page. The assessment requires you to draw our attention to the main features of your solution *clearly* and *concisely*. We will not search through lists of undocumented output trying to discover how your code is working.

Marking scheme

In what follows the *presentation* skill includes the appropriateness and depth of the description as well as its physical layout and display. Furthermore, solutions that are not FP (i.e. they use traditional/OO techniques including mutable state etc.) would represent an *insufficient* response to the problem.

Furthermore, if you choose a method which we have categorized as "low" difficulty then the top three categories will not apply to this solution. (Max for this part would therefore be 69).

If you choose a method which we have categorized as "medium" difficulty then the top two categories will not apply. (Max for this part would therefore be 79).

Does the answer overall represent a *reasonable* attempt to implement and describe the chosen method?

- Absolutely/yes/just about.
  - **No fault can be found** other than very minor (e.g. typographical) issues. Displays **exceptional** degree or originality and **exceptional** presentation skill. 90-100
  - An **outstanding** solution demonstrating a highly original and/or creative response and **excellent** presentation. 80-89
  - An extremely **well-developed** solution demonstrating depth of insight of the paradigm to construct a concise and elegant answer. It must be **very well presented**. 70-79
  - A **sound** solution demonstrating a **thorough grasp** and **appropriate application** of the underlying theory. The code does not have to be especially sophisticated so long as it represents a **sound** FP solution and it must solve **most**, if not all, of the required functionality. It must be **well presented**. 60-69
  - An **effective** solution. It should demonstrate **some** degree of insight into the **appropriate application** of relevant FP techniques but it may be a little inelegant in parts. It should at least solve **most** of the required functionality. It must be **clearly presented**. 50-59
  - A **basic** solution. It may only represent a **partial** solution. The code must apply FP concepts but these might be expressed in a **superficial** or particularly inelegant way. The **presentation should be sufficient** to convey a basic knowledge and understanding of the solution. 40-49
- No/not really. *In the following, even a nicely laid out presentation, i.e. layout, choice and organization of material on the page, cannot compensate for a lack of basic knowledge and/or a lack of appropriate analysis.*
  - The solution represents overall an **insufficient** response. The code contains particular **weaknesses** which may include, for example, non-FP elements. However, there must be FP elements within the code that are applied

appropriately. The presentation may be **weak**. 30-39

- A **poor** solution which falls **substantially short** of the expected level of development and application of knowledge. Nevertheless, there should still be **some evidence** of at least one FP technique applied appropriately within the code. The presentation may be **weak**. 20-29
- A **very poor** solution demonstrating **little understanding** of FP and its application. The solution most likely has **little adherence** to the task. 10-19
- An **insufficient** response. Displays virtually **no knowledge or understanding**. The work submitted may be **inappropriate** or **irrelevant** or **missing**. 0-9