COMP1811 (2022/23)	**Paradigms of Programming**	**Header ID**	**Contribution:** 20% of module
Module Leader/Moderator Yasmine Arafa/Andy Wicks	RESIT Practical Coursework 2 – Scheme Project		**Deadline Date** Friday 14/07/2023
This coursework should take an average student who is up to date with tutorial/lab work approximately **15 hours**. Feedback and grades are normally made available within 15 working days of the coursework deadline.			
Learning Outcomes: A. Understand the programming paradigms introduced and their applicability to practical problems. B. Apply appropriate programming constructs in each programming paradigm. C. Design, implement and test small-scale applications in each programming paradigm. D. Use appropriate tools to design, edit and debug programs for each paradigm.			

Plagiarism is presenting somebody else's work as your own. It includes copying information directly from the Web or books without referencing the material; submitting joint coursework as an individual effort; copying another student's coursework; stealing coursework from another student and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University. Please see your student handbook for further details of what is / isn't plagiarism.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using. Code snippets from open-source resources or YouTube must be acknowledged appropriately.

Your work will be submitted for electronic plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

- An **electronic copy** of your work for this coursework must be fully **uploaded by 23:30 on Friday 14/07/23.** Submissions must be made using the Scheme Project Upload link under "CW Details and Submission" on the Moodle page for COMP1811. **Your work will be capped if you fail to submit by the deadline**.
- For this coursework you must submit 2 files: a zip file containing the code files required to run **your Scheme project** and a screencast showing your code running, and a PDF file of **your report**, In general, any text in the report MUST NOT be an image (i.e. must not be scanned or a screenshot) and would normally be generated from other documents (e.g. MS Office using "Save As PDF"). **Marks will be deduced if you include images** instead of code or text. **There are limits on the file size**.
- Make sure that any files you upload are virus-free and not protected by a password or corrupted, otherwise, they will be treated as null submissions.
- Your work will be marked online, and feedback and a provisional grade will be available on Moodle. A news item will be posted when the feedback is available and when the grade is available in BannerWeb.
- All coursework must be submitted as above. Under no circumstances can they be accepted by academic staff.

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences. See http://www2.gre.ac.uk/current-students/regs

Coursework Regulations

- If you have Extenuating Circumstances, you may submit your coursework up to 10 working days after the published deadline without penalty, but this is subject to your claim being accepted by the Faculty Extenuating Circumstances Panel.
- Late submissions will be dealt with in accordance with University Regulations.
- Coursework submitted more than two weeks late may be given feedback but will be recorded as a non-submission regardless of any extenuating circumstances.
- Do not ask lecturers for extensions to published deadlines - they are not authorised to award an extension.

Please refer to the University Portal for further detail regarding the University Academic Regulations concerning Extenuating Circumstances claims.

Coursework Specification

Your overall task for the COMP1811 Scheme Project Coursework is to design and develop a small program in Scheme. You are expected to work individually to develop this project.

Implementing the program will give you experience in the development of an application using Functional Programming, where you are expected to design, implement, and use your own code from scratch or a template. Your code development is the culmination of all that you have learned and all your hard work for part two of this module (Functional Programming).

Your solutions are expected to be integrated into the code template provided. Please, stick to the template given to you and complete the Scheme definitions with your own code.

It is recommended to switch off your mobile. Programming requires mindfulness.

Please read the entire coursework specification. Reading and understanding take more time than coding.

Scenario

You are programming your second App in Scheme: "*Messaging System*". See video at Scheme-Resit-Demo.

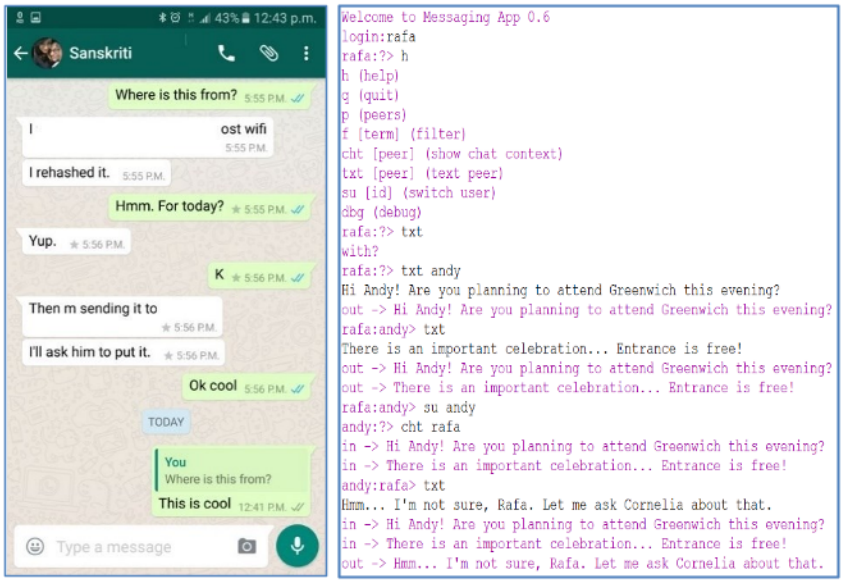

Figure 1 Simulating WhatsApp© with Dr. Racket REPL.

Description of Coursework

You are to simulate a messaging system. To save the huge complexity gap between Graphical User Interfaces and limitations of Racket REPL, we must do some assumptions in the running app:

- The prompt **<u1:u2> ->** states u1 chatting with peer u2.
- To text a message to any peer, enter **txt [peer]**.
- To access a peer's chat context, enter **cht [peer].**
- To switch the chat user, enter **su [user].**
- To list your peers, enter **p.**
- To filter all chat messages containing a topic-term, enter **f [term].**

Fundamentals.

The rationale underlying the system is understanding the state as a nested dictionary. **dbg** command should yield the following screenshot:

```
cornelia> dbg
rafa ->
        andy ->
                out -> Hi andy! Are you attending Greenwich tomorrow?
                out -> A party is to be held in KW premises
                in -> I'm not sure yet. Let me ask Cornelia about it
andy ->
        rafa ->
                in -> Hi andy! Are you attending Greenwich tomorrow?
                in -> A party is to be held in KW premises
                out -> I'm not sure yet. Let me ask Cornelia about it
        cornelia ->
                out -> Hi cornelia! Fancy a party a Greenwich with rafa?
                in -> No, I'm leaving London. Anyway Thanks!
cornelia ->
        andy ->
                in -> Hi cornelia! Fancy a party a Greenwich with rafa?
                out -> No, I'm leaving London. Anyway Thanks!
```

- The first level is indexed by the users of system (**rafa,andy,cornelia**).
- Private chats held peer-to-peer are indexed at second level (e.g,(**rafa,cornelia**) have been addressed by **andy**).
- List of messages are the values of every addressee, being each message a pair with a qualifier (**in|out**) to the content of it. Mind how qualifiers are relative to origin/destination of the message.

You must program using only *pure, side-effects free functions, using recursion*[1] *or high-order programming.* In practical terms, you cannot use **set!, printf, read, for-each, begin** or any other side effect functions in Scheme. Function with side effects conventionally have an exclamation mark (!) as the final character of the function name.

Task 0

Familiarise yourself with the given code.

1) Acquire the source code from **sn-scheme-resit-template** on Moodle.
2) Download, unzip and run it in **DrRacket (**open sn-app.rkt and run).
3) Locate the file and the functions you are expected to implement (file **sn-utils.rkt**).
4) Please DO NOT alter the rest of files (**sn-io.rkt,sn-console.rkt**). You may read but not modify them. They contain some auxiliary routines as well as the base control flow for the entire application, calling functions implemented by you at 3)**.**

[1] The use of **loop** construct is forbidden.

Task 1

Here you have a description of signatures of functions supporting the commands above.

i. sn-users (10 marks).

Following the rationale above, assuming **system** properly encoded and **uid** (as symbol) returns a list of users **uid** has held a chat with.

```
(define (sn-users system uid)  null)
```

ii. sn-txt (30 marks)

Ditto. **u1** and **u2** are the source and target of message **txt** as string. It simulates sending a message according to the above rationale.

```
(define (sn-txt system u1 u2 txt) null)
```

iii. sn-chat (10 marks)

Ditto. **who** is the peer whose chat's context we want to access. The result is a local dictionary according to the above rationale.

```
(define (sn-chat system uid who) null)
```

iv. sn-filter (30 marks)

Ditto. **uid** is the user whose messages we want to filter containing the given **topic** (as a string).

```
(define (sn-filter system uid topic)  null)
```

Task 2

Write a **brief report** describing the main FP topics you used to implement each function (i.e., *recursion, high-order, lists, union, intersection, dictionaries, meta-programming…)* and your design decisions. Please use the report template provided on Moodle.

Upload the folder containing the **Scheme code you developed (.rkt files), your final report as a pdf file, and a 7–10-minute screencast (explaining the code and showing your system running)** to Moodle under the Scheme Project Upload link under "CW Details and Submission" **by 23:30 on 14/07/2023**.

You may fail this assessment if you fail to submit a screencast.

Marking Scheme (mark breakdown)

Routine	Marks
sn-users	10
sn-chat	10
sn-filter	30
sn-txt	30
Screencast	10
Report	10

Grading Criteria

Grading Criteria Judgement for the items in the marking scheme will be made based on the following criteria. To be eligible for the mark in the left-hand column you should at least achieve what is listed in the right-hand column. Note that you may be awarded a lower mark if you do not achieve all the criteria listed. For example, if you achieve all the criteria listed for a 2:1 mark but have a poor report then your mark might be in the 2:2 range or lower.

> 80% *Exceptional*	• Completed functionality implemented and tested to an outstanding standard or above (all required features implemented, no obvious bugs, outstanding code quality, Scheme language features accurately applied). • Able to show outstanding, thorough knowledge and systematic understanding of functional programming concepts and Scheme language features in the code and the report. • Overall report – outstanding (required sections completed accurately and clearly, easy to read, structured and coherent and insightful arguments for design justification and use of Scheme language). • Evaluation section of report – outstanding (insightful points made relevant to development (system produced), productivity, errors, learning, and time management; thorough introspection, realistic and insightful reflection).
70-79% *Excellent*	• Completed functionality implemented and tested to an excellent standard (required features implemented, no obvious bugs, excellent code quality, Scheme language features accurately applied). • Able to show excellent, detailed knowledge and systematic understanding of functional programming concepts and Scheme language features in the code and in the report. • Overall report – excellent (required sections completed accurately and clearly, easy to read, well justified design decisions and clear argument, comparative reasoning).

	• Evaluation section of report – excellent (interesting points made relevant to development (system produced), productivity, errors, learning, and time management; detailed introspection and interesting reflections).
60-69% *Very Good*	• Completed functionality implemented and tested to a very good standard (required features implemented, few if any bugs, very good code quality, very good attempt using Scheme language features, may contain some design flaws). • Able to show very good knowledge and systematic understanding of functional programming concepts and Scheme language features in the code and in the report. • Overall report – very good (required sections completed accurately and clearly, good argument for design justification and use of Scheme). • Evaluation section of report – very good (very good points made relevant to at least three out of the following: development (system produced), productivity, errors, learning, and time management; introspection shows some reasonable thought).
50-59% *Good*	• Completed functionality implemented and tested to a good standard (required features implemented, few if any bugs, good attempt using Scheme language features, contains some design flaws). • Able to show good knowledge and mostly accurate systematic understanding of functional programming concepts and Scheme language features in the code and in the report. • Overall report – good (required sections completed accurately, mostly clear, some reasonably balanced argument for design justification and use of Scheme). • Evaluation section of report – good (addresses points relevant to at least three out of the following: development (system produced), productivity, errors, learning, and time management; limited introspection).
45-49% *Satisfactory*	• Completed functionality implemented and tested to a good standard (some features implemented, few if any bugs, acceptable attempt using Scheme language features, contains some design flaws). • Able to show satisfactory knowledge of functional programming concepts and Scheme language features in the code and in the report. • Overall report – acceptable (required sections completed, mostly accurate and clear, superficial justification for design choices and use of Scheme). • Evaluation section of report – acceptable (addresses points relevant to at least two out of the following: development (system produced), productivity, errors, learning, time management, mostly descriptive; superficial introspection).
40-45% *Satisfactory*	• Completed functionality implemented and tested to a reasonable standard (at least 2 required features implemented with a few minor bugs, acceptable attempt using Scheme language features, contains some design flaws). • Able to show fairly satisfactory knowledge of functional programming concepts and Scheme language features in the code and in the report. • Overall report – acceptable (required sections completed, mostly accurate, clumsy language, descriptive account of design choices and use of Scheme). • Evaluation section of report – acceptable (addresses points relevant to at least one out of the following matters: development (system produced), productivity, errors, learning, time management, mostly descriptive; superficial introspection).

35-39% *Fail*	• Good attempt at some features although maybe buggy with some testing. • Confused knowledge of functional programming concepts and Scheme language features in the code and in the report. • Overall report – mostly completed to an acceptable standard. Some sections are confused. • Evaluation section of report – mostly descriptive but showing some thought.
<35% *Fail*	• Little or no attempt at features or very buggy with little or no testing. • Not able to show satisfactory knowledge functional programming concepts and Scheme language features in the code and in the report. • Overall report – mostly in-completed, at an un-acceptable standard or missing. • Evaluation section of report – descriptive showing a lack of thought or missing.