Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 4: Functional and Logic Programming

September 5, 2023 (Updated September 7, 2023)

1 Objectives

This practical aims to achieve the following general learning objectives:

- To consolidate the concepts covered in Chapter 15 of the prescribed textbook.
- To consolidate the concepts covered in Chapter 16 of the prescribed textbook.

2 Plagiarism Policy

Plagiarism is a serious form of academic misconduct. It involves both appropriating someone else's work and passing it off as one's own work afterwards. Thus, you commit plagiarism when you present someone else's written or creative work (words, images, ideas, opinions, discoveries, artwork, music, recordings, computer-generated work, etc.) as your own. Note that using material produced in whole or part by an AI-based tool (such as ChatGPT) also constitutes plagiarism. Only hand in your own original work. Indicate precisely and accurately when you have used information provided by someone else. Referencing must be done in accordance with a recognised system. Indicate whether you have downloaded information from the Internet. For more details, visit the library's website: `http://www.library.up.ac.za/plagiarism/`.

3 Submission Instructions

Upload your practical-related source code file to the appropriate assignment upload slots on the ClickUP course page. Upload only the source code files for the Scheme and Prolog implementations. Name the Scheme implementation file `s99999999.scm`, and name the Prolog implementation file `s99999999.pl`. In both cases, `99999999` is your student number. Multiple uploads are allowed, but only the last one will be marked. The submission deadline is **Tuesday, 3 October 2023, at 12:00**.

4 Background Information

You will be implementing your first task in Scheme and your second task in Prolog. Please refer to the background information in practicals 2 and 3 for further information on how to write and interpret Scheme and Prolog programs. DrRacket 8.9 using the sicp language collections will be used to assess your Scheme submission. SWI-Prolog version 9.0.4 will be used to assess your Prolog submission.

5 Practical Tasks

5.1 Task 1: Scheme

You must once again use the DrRacket interpreter using the sicp language collections to run and test your submission for this task. For further information on the DrRacket interpreter, please see the specification for Practical 2.

Once again, note that you may only use the following functions and language features in your programs, and that failure to observe this rule will result in all marks for this task being forfeited:

Function construction: `lambda`, `define`

Binding: `let`

Arithmetic: `+`, `-`, `*`, `/`, `abs`, `sqrt`, `remainder`, `min`, `max`

Boolean values: `#t`, `#f`

Equality predicates: `=`, `>`, `<`, `>=`, `<=`, `even?`, `odd?`, `zero?`, `negative?`, `eqv?`, `eq?`

Logical predicates: `and`, `or`, `not`

List predicates: `list?`, `null?`

Conditionals: `if`, `cond`, `else`

Quoting: `quote`, `'`

Evaluation: `eval`

List manipulation: `list`, `car`, `cdr`, `cons`

Input and output: `display`, `printf`, `newline`, `read`

You may also define any additional helper functions that you require.

Write a function named `getDivisors`, which receives two parameters. The first parameter is an integer value, for which you want to find divisors (i.e. you want to find values that divide perfectly into this parameter). The second parameter is a list containing non-zero integer values that must be searched for divisors. The `getDivisors` function should yield a list containing only the divisors found in the list provided as the second parameter.

For example, the function application

```
(getDivisors 10 '())
```

should yield an empty list, because the second parameter contains no divisors of 10. As another example, the function application

```
(getDivisors 10 '(20 50 60))
```

should also yield an empty list, because 20, 50, and 60 are all not divisors of 10. As a final example, the function application

```
(getDivisors 10 '(1 20 30 2 5 40 10 60))
```

should yield the list `(1 2 5 10)` because these numbers are all divisors of 10 (note that 10 is a divisor of itself), while 30, 40 and 60 are not divisors of 10.

5.2 Task 2

You must once again use the SWI-Prolog interpreter to run and test your submission for this task. For further information on the SWI-Prolog interpreter, please see the specification for Practical 3.

Once again, take note of the following requirements for your submission:

- **Note that you may only use the simple constants, variables, list manipulation methods, and built-in predicates discussed in the textbook and slides. You may NOT use any more complex predicates provided by the Prolog system itself. In other words, you must write all your own propositions. Failure to observe this rule will result in all marks for a task being forfeited.**
- **You may implement and use the propositions defined in the textbook and slides (e.g. `member`, `append`, and `reverse`). Note that you must provide the implementation for any of these propositions in your source file. Also note that there are some built-in propositions that correspond to the propositions defined in the textbook, which you may not use.**
- You may implement helper propositions if you find them necessary.

Write a Prolog proposition named `elementAt` that has three parameters. The first parameter is an integer constant, the second parameter is a simple list (i.e. a list containing only constants), and the third parameter is a constant (i.e. a symbolic atom or an integer). The proposition defines the third parameter to be the constant located in the list at the position denoted by first integer parameter, where the first position is 1. To illustrate the use of the `elementAt` proposition, consider the following queries and responses:

```
?- elementAt(1, [a, b, c], a).
true.

?- elementAt(3, [a, b, c], z).
false.

?- elementAt(5, [a, b, c], c).
false.

?- elementAt(2, [a, b, c], X).
X = b.
```

Hint: You will have to use Prolog's support for simple arithmetic. Refer to the textbook and slides for more information on how simple arithmetic is supported in Prolog.

6 Changes Since Last Version

Clarified that parameter list in Task 1 can be assumed to not contain zero values.

7 Marking

Each of the tasks will count 5 marks for a total of 10 marks. Submit the Scheme and Prolog implementations to the appropriate assignment upload slots. Do not upload any additional files other than your source code. Both the implementation and the correct execution of the programs will be taken into account. **You will receive zero for a task that uses a language feature you are not allowed to use**. Your program code will be assessed during the practical session in the week of **Monday, 2 October 2023**.