Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 2: Functional Programming

August 15, 2023

1 Objectives

This practical aims to achieve the following general learning objectives:

- To gain and consolidate some experience writing functional programs in Scheme;
- To consolidate the concepts covered in Chapter 15 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 slot on the ClickUP course page. You will be implementing all the practical's tasks in the same file. Name this file `s99999999.scm`, where `99999999` is your student number. Multiple uploads are allowed, but only the last one will be marked. The submission deadline is **Tuesday, 29 August 2023, at 12:00**.

4 Background Information

For this practical, you will be writing functional programs in Scheme:

- You will have to use the DrRacket 8.8 IDE using the sicp language collections, which is installed on the Windows computers in the Informatorium. Instructions for installing sicp are available at `https://docs.racket-lang.org/sicp-manual/Installation.html`. You should write your function implementations in DrRacket. Note that your program must always start with the line:

  ```
  #lang sicp
  ```

  You can then run the source file by clicking on the "Run" icon at the top of the screen. You must include the code to test your functions at the end of your program (see below for some sample inputs).

- A reference manuals for MIT/GNU Scheme [1] has been uploaded on the course website. Please also refer to chapter 15 of the textbook, and the slides that have been uploaded for that chapter. Also note that the sicp implementation of Scheme uses lowercase letters for built-in function names (e.g. `odd?` rather than `ODD?`), and doesn't provide all the functions discussed in the slides and textbook (e.g. `<>` is not provided).

- **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 a 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`

5 Practical Tasks

For this practical, you will need to explore and implement functional programming concepts, including list processing and the use of recursive functions. This practical consists of three tasks, and all three tasks should be implemented in a single source code file.

5.1 Task 1

Write a function named `pyramidVolume`, which receives two parameters. The first parameter is a numeric atom representing the length of the side of the square base, while the second parameter is a numeric atom representing the height of the apex above the base. The function should yield the volume of the pyramid defined by the provided length and height, computed as:

$$V = \frac{1}{3}l^2h$$

where l is the length of the side of the square base, and h is the height of the apex above the base. The function should yield zero if either l or h is negative. Use a `let` (not a `define`) to define a name for the value of the result of the calculation 1/3.

For example, the function application

```
(pyramidVolume 3.2 4.8)
```

should yield a result of approximately `16.384`. **Only use the prescribed language features and functions provided above**.

5.2 Task 2

Write a function named `getPositiveOddNumbers`, which receives one parameter. The parameter is a simple list (i.e. a list containing only atoms). You may assume that the list contains only numeric atoms. The function should yield a list containing only the positive (non-zero) odd values contained in the parameter list.

For example, the function application

```
(getPositiveOddNumbers '())
```

should yield an empty list because there are no elements in the parameter list. As another example, the function application

```
(getPositiveOddNumbers '(-3 4 0 5 8 3))
```

Should yield the list `(5 3)` because `5` and `3` are the only positive odd values in the list (`-3` and `0` are not positive, and `4` and `8` are positive even values).

To implement the `getPositiveOddNumbers` function, you will have to recursively traverse the parameter list, and construct a result list. **Only use the prescribed language features and functions provided above**.

5.3 Task 3

Write a function named `doubleEveryOddElement`, which receives one parameter. The parameter is a simple list (i.e. a list containing only atoms). You may assume that the list contains only numeric atoms. The function should yield a list containing all the values in the original list, with all the values at odd positions doubled (assuming that the first element in the list is at position 1).

For example, the function application

```
(doubleEveryOddElement '())
```

should yield an empty list, because the parameter list contains no values. As another example, the function application

```
(doubleEveryOddElement '(4))
```

should also yield the list `(8)`, because the only value contained in the list is at position 1, which is an odd position. The double of `4` is `8`. As a final example, the function application

```
(doubleEveryOddElement '(4 3 5 2))
```

should yield the list `(8 3 10 2)` because `4` is at position 1 in the list, and `5` is at position 3 in the list. The double of `4` is `8`, and the double of `5` is `10`.

To implement the `doubleEveryOddElement` function, you will have to recursively traverse the parameter list, and build up a result list. **Only use the prescribed language features and functions provided above**.

Hint: Consider writing multiple functions for this task, where each function performs a different part of the required processing on the parameter list.

6 Marking

Each of the tasks will count 5 marks for a total of 15 marks. Submit all three tasks implemented in the same source code file. Do not upload any additional files other than your source code. Both the implementation and the correct execution of the functions 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, 28 August 2023**.

References

[1] Chris Hanson and the MIT Scheme Team. MIT/GNU Scheme reference manual. Technical report, Massachusetts Institute of Technology, 2018.