Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 2: Functional Programming

August 16, 2022

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

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.ais.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the `Library` quick link, and then click the `Plagiarism` link). If you have any form of question regarding this, please consult the lecturer, to avoid any misunderstanding. Also note that the principle of code re-use does not mean that you should copy and adapt code to suit your solution. Note that all assignments submitted for this module implicitly agree to this plagiarism policy, and declare that the submitted work is the student's own work. Assignments will be submitted to a variety of plagiarism checks. Any typed assignment may be checked using the Turnitin system. After plagiarism checking, assignments will not be permanently stored on the Turnitin database.

3 Submission Instructions

Upload your practical-related source code file to the appropriate assignment upload slots 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, 30 August 2022, at 23:00**.

4 Background Information

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

- You will have to use a Scheme interpreter called GNU Guile. Guile version 2.2.3 will be used to assess your submissions. Please see the Linux `man` page on the `guile` command for information on how to use the GNU Guile interpreter. You should write your function implementations as source code in an ASCII text file. You can then interpret the source file by using the `guile` command with the `-l` switch. This will present you with an interactive prompt where you can type code to test your functions (see below for some sample inputs). Pressing `ctrl+d` will leave the interactive prompt (you may have to press `ctrl+d` several times if you've entered several lines of input).
- A reference manuals for MIT/GNU Scheme [1] has been uploaded on the course website. Additionally, the GNU Guile project homepage [2] maintains a list of many useful resources. Please also refer to chapter 15 of the textbook, and the slides that have been uploaded for that chapter. Also note that GNU Guile 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, namely 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 `coneVolume`, which receives two parameters. Both parameters are numeric values, the first representing the radius of the cone's base, and the second representing the height of the cone. The function should yield the volume of the cone with the provided radius and height. Look up the formula for calculating this volume yourself. Use a `let` (not a `define`) to bind the value of $\frac{22}{7}$ to a name representing π for use in the volume formula. The function should yield a zero volume if either the radius or the height are negative.

For example, the function application

```
(coneVolume 3.2, 1.8)
```

should yield a result of `19.309714285714286` (at least approximately). **Only use the prescribed language features and functions provided above**.

5.2 Task 2

Write a function named `countNegatives`, which receives one parameter. The parameter is a list containing integer values that must be searched for negative values (exclusing zero). The `countNegatives` function should yield a count of the number of negative values found in the list provided as the parameter.

For example, the function application

```
(countNegatives '())
```

should yield `0`, because the parameter contains no negative values. As another example, the function application

```
(countNegatives '(20 50 60))
```

should also yield `0`, because 20, 50, and 60 are all non-negative values. As a final example, the function application

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

should yield `4` because -1, -30, -5, and -60 are all negative values, while 20, 2, 40, and 10 are non-negative.

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

5.3 Task 3

Write a function named `getEveryOddElement`, which receives one parameter. The parameter is a simple list (i.e. a list containing only atoms) from which you want to extract all the values at odd numbered positions (assuming that the first element in the list is at position 1). The `getEveryOddElement` function should yield a list containing all the values extracted from the parameter list.

For example, the function application

```
(getEveryOddElement '())
```

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

```
(getEveryOddElement '(a))
```

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

```
(getEveryOddElement '(a b c d))
```

should yield the list `(a c)` because `a` is at position 1 in the list, and `c` is at position 3 in the list.

To implement the `getEveryOddElement` function, you will have to recursively traverse the parameter list, and build up a 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. 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 marked offline by the teaching assistants. Make sure that you upload your complete program code to the appropriate assignment upload slot, and include comments at the top of your program source file, explaining how your program should be executed, as well as any special requirements or limitations that your program has. Do not upload any additional files other than your source code.

References

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

[2] GNU Guile project team. GNU Guile project homepage. Access at `https://www.gnu.org/software/guile/`.