CP472: Assignment 4
Total Points = 35

In this programming assignment, you will explore the fundamental concepts of the Scheme programming language through a series of exercises. You will practice basic syntax, list manipulation, recursion, and functional programming paradigms. The goal is to gain a deeper understanding of Scheme's unique features and to apply functional programming techniques to solve various problems. This assignment is designed to help you develop your skills in Scheme and to appreciate the elegance and power of functional programming.

Tasks:

1. Write a Scheme program that calculates the simple interest for a given principal amount, rate of interest, and time in years. The formula for calculating simple interest is:

$$\text{Simple Interest} = \frac{\text{Principal} \times \text{Rate} \times \text{Time}}{100}$$

   where
   - Principal is the initial amount of money.
   - Rate is the annual interest rate (in percentage).
   - Time is the time period for which the interest is calculated (in years).

   Your program should take the principal amount, rate of interest, and time as inputs and print the calculated simple interest.

2. Implement a function that takes a list of numbers and returns the sum of squares of those numbers using map and fold (or reduce) functions.
3. Write a function that filters a list of numbers using filter function to keep only the even ones and then accumulates their product.
4. Write a Scheme program that takes a list of numbers as input and sorts them in ascending order using the bubble sort algorithm.
5. Write a Scheme program that takes two matrices as input and outputs their product. Assume the matrices are represented as lists of lists.
6. Create a higher-order function named apply-to-each that takes two arguments: a function f and a list lst. The function f should be a unary function, meaning it takes a single argument and returns a value. The apply-to-each function should iterate over each element of the list lst, apply the function f to each element, and collect the results into a new list. Finally, apply-to-each should return this new list containing the results of applying f to each element of the original list lst.

For example, if f is a function that doubles its input, and lst is the list (1 2 3 4 5), then apply-to-each should return the list (2 4 6 8 10) after applying f to each element of lst.

7. Write a Scheme program that solves the Towers of Hanoi problem for a given number of disks. The Towers of Hanoi is a classic puzzle that consists of three pegs and a number of disks of different sizes, which can slide onto any peg. The puzzle starts with the disks in a neat stack in ascending order of size on one peg, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another peg, obeying the following rules:

   - Only one disk can be moved at a time.
   - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty peg.
   - No disk may be placed on top of a smaller disk.

   Your program should take the number of disks as input and print the sequence of moves required to transfer all the disks from the first peg to the third peg, using the second peg as an auxiliary. Each move can be represented as a tuple (source, destination), where source and destination are the numbers of the pegs involved in the move (e.g., (1, 3) would represent moving a disk from peg 1 to peg 3).

   Here's an outline of a possible solution using recursion:

   - If there is only one disk, move it directly from the source peg to the destination peg.
   - For more than one disk, recursively move all but the bottom disk from the source peg to the auxiliary peg.
   - Move the bottom disk from the source peg to the destination peg.
   - Finally, recursively move the disks from the auxiliary peg to the destination peg.

   The program should provide clear output showing each move, and it should work for any number of disks. The solution should demonstrate an understanding of recursion and how it can be used to solve complex problems in a simple and elegant way.

Marking Scheme:

All questions contain equal marks.

Deliverables

Submit a zip file containing:

- The source code files.
- Include a README file mentioning which version and interpreter of scheme you have used.