CS305 – Programming Languages
Spring 2024-2025

HOMEWORK 4

**Warming Up in Scheme Language**

**Due date: May 12, 2025 @ 23:55**

NOTE

Only SUCourse submission is allowed. No submission by e-mail. Please see the note at the end of this document for late submission policy.

# 1 Introduction

In this homework you will implement various Scheme procedures that will hopefully give you a better grasp on the language.

# 2 Anapali

In this homework, you will implement some procedures related to the following concepts on sequences.

- **Palindrome**: A sequence of symbols $\sigma$ is called a palindrome if $\sigma$ is the same whether you read it from left to right or from right to left.
  - For example, the empty sequence, and the sequences `a`, `aaaa`, `abaaba`, `neveroddoreven` are all palindromes.
- **Anagram**: A sequence $\sigma$ is an anagram of another sequence $\sigma'$ if the symbols of $\sigma$ can be shuffled to obtain $\sigma'$.
  - For example, `abcb` is an anagram of `bacb`, or `aaabbb` is an anagram of `ababab`. Similarly, `army` is an anagram of `mary`.
- **Anapali**: A sequence $\sigma$ is an anapoli if $\sigma$ is an anagram of a palindrome. In other words, $\sigma$ is an anapoli if it is possible to rearrange the symbols in $\sigma$ to obtain a palindrome.
  - For example, `prepeellers` is an anapoli since it is an anagram of `lepersrepel`, which is a palindrome.

# 3 Error Messages

In this homework, you will see that we ask you to display error messages under some conditions.**You should NOT use the ERROR procedure of Scheme**, you have to print the ERROR message using the display procedure of Scheme. All your error messages should be in the form:

ERROR305: OPTIONAL ERROR MESSAGE

This means that all your error messages must start with the keyword ERROR305 followed by a colon followed by an optional error message of your choice for each different error.

# 4 Scheme procedures to be implemented

You will implement the following procedures in Scheme. We will represent the sequences as lists of Scheme symbols where each symbol will be a single item. For example the palindromic sequence aba will be represented as the Scheme list of symbols (a b a) with three elements where the first element is the symbol a, the second element is the symbol b and the third element is again the symbol a. We will never use a symbol of length more than one (e.g. a symbol like ab will not be used). In the Scheme procedures to be implemented below, whenever we talk about a "sequence", please assume that it is such a list. The procedures to be implemented are really easy. You can define helper procedures to be used in your code. You can also use the procedure symbol-length provided below. However, do not use any built-in procedure other than the ones we've covered in our lecture notes. List of procedures to be implemented shown in Chapter 5.

```
;- Procedure: symbol-length
;- Input : Takes only one parameter named inSym
;- Output : Returns the number of items in the symbol.
;  If the input is not a symbol, then it returns 0.
;- Examples :
; (symbol-length 'a) ------> returns 1
; (symbol-length 'abc) ----> returns 3
; (symbol-length 123) -----> returns 0
; (symbol-length '(a b)) --> returns 0
(define symbol-length
    (lambda (inSym)
        (if (symbol? inSym)
            (string-length (symbol->string inSym))
        0
        )
    )
)
```

# 5 Functions

You need to define various functions that include the combination of the rules described above. Define each function explained below separately.

```
;- Procedure: sequence?
;- Input : Takes only one parameter named inSeq
;- Output : Returns true if inSeq is a list of symbols where each
; symbol is of length 1
;- Examples :
; (sequence? '(a b c)) ----> returns true
; (sequence? '()) ---------> returns true
; (sequence? '(aa b c)) ---> returns false since aa has length 2
; (sequence? '(a 1 c)) ----> returns false since 1 is not a symbol
; (sequence? '(a (b c))) --> returns false since (b c) is not a symbol
; (sequence? 'a) --> returns false since a is not a list
(define sequence?
    (lambda (inSeq)
    ...
    )
)
```

```
;- Procedure: same-sequence?
;- Input : Takes two sequences inSeq1 inSeq2 as input
;- Output : Returns true if inSeq1 and inSeq2 are sequences and they are the
; same sequence.
; Returns false if inSeq1 and inSeq2 are sequences but they are not the same sequence.
; If inSeq1 is not a sequence or inSeq2 is not a sequence, it produces an error.
;- Examples :
; (same-sequence? '(a b c) '(a b c)) --> return true
; (same-sequence? '() '()) ------------> return true
; (same-sequence? '(a b c) '(b a c)) --> returns false
; (same-sequence? '(a b c) '(a b)) ----> returns false
; (same-sequence? '(aa b) '(a b c)) ---> produces an error
(define same-sequence?
    (lambda (inSeq1 inSeq2)
    ...
    )
)
```

```
;- Procedure: reverse-sequence
;- Input : Takes only one parameter named inSeq
;- Output : It returns the reverse of inSeq if inSeq is a sequence. Otherwise
;  it produces an error.
;- Examples :
; (reverse-sequence '(a b c)) --> returns (c b a)
; (reverse-sequence '()) -------> returns ()
; (reverse-sequence '(aa b)) ---> produces an error
(define reverse-sequence
    (lambda (inSeq)
    ...
    )
)

;- Procedure: palindrome?
;- Input : Takes only one parameter named inSeq
;- Output : It returns true if inSeq is a sequence and it is a palindrome.
;  It returns false if inSeq is a sequence but not a palindrome.
;  Otherwise, i.e. if inSeq is not  a sequence it gives an error.
;- Examples :
; (palindrome? '(a b a)) --> returns true
; (palindrome? '(a a a)) --> returns true
; (palindrome? '()) -------> returns true
; (palindrome? '(a a b)) --> returns false
; (palindrome? '(a 1 a)) --> produces an error
(define palindrome?
    (lambda (inSeq)
    ...
    )
)


;- Procedure: member?
;- Input : Takes a symbol named inSym and a sequence named inSeq
;- Output : It returns true if inSeq is a sequence and inSym is a symbol
;  and inSym is one of the symbols in inSeq. It returns false if inSeq is a sequence
;  and inSym is a symbol but inSym is not one of the symbols in inSeq.
;  It produces an error if inSeq is not a sequence or if inSym is not a symbol.
;- Examples :
; (member? 'b '(a b c)) ---> returns true
; (member? 'd '(a b c)) ---> returns false
; (member? 'd '()) --------> returns false
; (member? 'aa '(a b c)) ---> produces an error
; (member? 5 '(a 5 c)) ----> produces an error
; (member? 'd '(aa b c)) --> produces an error
```

```
(define member?
    (lambda (inSym inSeq)
    ...
    )
)


;- Procedure: remove-member
;- Input : Takes a symbol named inSym and a sequence named inSeq
;- Output : If inSym is a symbol and inSeq is a sequence and inSym is one of
;  the symbols in inSeq, then remove-member returns the sequence
;  which is the same as the sequence inSeq, where the first occurrence of
;  inSym is removed. For any other case, it produces an error.
;- Examples :
; (remove-member 'b '(a b b c b)) --> returns (a b c b)
; (remove-member 'd '(a b c)) ------> produces an error
; (remove-member 'b '()) -----------> produces an error
; (remove-member 'b '(a b 5 c)) ----> produces an error
; (remove-member 5 '(a b c)) -------> produces an error
(define remove-member
    (lambda (inSym inSeq)
    ...
    )
)

;- Procedure: anagram?
;- Input : Takes two sequences inSeq1 and inSeq2 as paramaters.
;- Output : If inSeq1 and inSeq2 are both sequences and if inSeq1 is an
;  anagram of inSeq2, then anagram? returns true.
;  If inSeq1 and inSeq2 are both sequences and but if inSeq1 is not an
;  anagram of inSeq2, then anagram? evaluates to false.
;  If inSeq1 is not a sequence or if inSeq2 is not a sequence then anagram?
;  produces an error.
;- Examples :
; (anagram? '(m a r y) '(a r m y)) --> returns true
; (anagram? '() '()) ----------------> returns true
; (anagram? '(a b b) '(b a a)) ------> returns false
; (anagram? '(a b b) '()) -----------> returns false
; (anagram? '(a bb) '(a bb)) --------> produces an error
; (anagram? 5 '(a b b)) -------------> produces an error
(define anagram?
    (lambda (inSeq1 inSeq2)
    ...
    )
)
```

```
;- Procedure: unique-symbols
;- Input : Takes a sequence named inSeq
;- Output : Returns a sequence containing only the unique symbols from inSeq in the
;  order of the first occurrence of the symbols.
;- If inSeq is not a sequence, it produces an error.
;- Examples :
; (unique-symbols '(a b a c)) ----------> returns (a b c)
; (unique-symbols '(a b a b c b a)) ----------> returns (a b c)
; (unique-symbols '()) --------------------> returns ()
; (unique-symbols '(a a a)) --> returns (a)
; (unique-symbols '(a b 5 c))  ------> produces an error
(define unique-symbols
    (lambda (inSeq)
    ...
    )
)

;- Procedure: count-occurrences
;- Input : Takes a symbol named inSym and a sequence named inSeq
;- Output : Returns the number of times inSym appears in inSeq.
;- If inSeq is not a sequence or inSym is not a symbol, it produces an error.
;- Examples :
; (count-occurrences 'a '(a b a c) ----------> returns 2
; (count-occurrences 'b '(a b a c)) --------------------> returns 1
; (count-occurrences 'd '(a b a c))) --> returns 0
; (count-occurrences 5 '(a b a c))  ------> produces an error
; (count-occurrences 'a '(aa (b a)))  ------> produces an error

(define count-occurrences
    (lambda (inSym inSeq)
    ...
    )
)

;- Procedure: anapoli?
;- Input : Takes a sequence inSeq1 as parameter.
;- Output : If inSeq1 is not a sequence, it produces an error.
;  When inSeq1 is sequence, it returns true, if inSeq1 is an anagram
;  of any palindrome. In other words, if it is possible to rearrange
;  the symbols in inSeq1 to obtain a palindrome sequence, it returns true.
;  Otherwise, it returns false.
;- Examples :
; (anapoli? '(a b b)) ----------> returns true since we can obtain bab
; (anapoli? '()) --------------------> returns true
; (anapoli? '(d a m a m)) --> returns true since we can obtain madam
```

```
; (anapoli? '(a b b c)) ------> returns false
; (anapoli? '(aa bb b c)) ------> produces an error
; (anapoli? '(5 a bb)) --------------------> produces an error

(define anapoli?
    (lambda (inSeq1)
    ...
    )
)
```

# 6 Rules

- **Important**: **You are not allowed to use let*** while defining functions described in Section 5. The reason is that you should get used to the functional way of thinking while using a functional programming language.

# 7 How to Submit

Submit your Scheme file named as username-hw4.scm where username is your SU-Course username. In this file you must have scheme procedures defined with the names given in Section 5.

You can have additional procedures that you use within the code, but we will only test/grade by using the procedures given above.

# 8 Notes

- **Important**: Name your files as you are told and **don't zip them.** [-10 points otherwise]
- **Important**: The golden is shared as the file

  `onur.dogan/cs305/hw4-golden.com`

  on `cs305.sabanciuniv.edu`

  You can access the functions to be implemented by loading this file as

  `(load "hw4-golden.com")`

  We have provided the tests used in this document, along with their function calls, in the "hw4-examples.txt" file located in the same directory, so that you can easily try them.

- **Important: Since this homework is evaluated automatically make sure your output is exactly as it is supposed to be. Just include your procedures in the submission file. Some of the points that we can think of are:**
  - **Not displaying the error message**
  - **Putting some test codes that calls the procedures at the end of the procedures or file**
- If you are not sure about your outputs you can compare your outputs with the outputs given by the golden.
- You may get help from our TA or from your friends. However, **you must implement the homework by yourself**.
- Start working on the homework immediately.
- If you develop your code or create your test files on your own computer (not on cs305.sabanciuniv.edu), there can be incompatibilities once you transfer them to the cs305 machine. Since the grading will be done automatically on the cs305 machine, we strongly encourage you to do your development on the cs305 machine, or at least test your code on the cs305 machine before submitting it. If you prefer not to test your implementation on the cs305 machine, this means you accept to take the risks of incompatibilities. Even if you may have spent hours on the homework, you can easily get 0 due to such incompatibilities.

## LATE SUBMISSION POLICY

Late submission is allowed subject to the following conditions:

- Your homework grade will be decided by multiplying what you get from the test cases by a "submission time factor (STF)".
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don't lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse's timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.