Program 6: Scheme

A Scheme template for **p6.scm** defines some *lists* with which you can experiment; all you need to do is add the definitions of five functions, the first four of which *must be* **recursive.**

- odds
- evenrev
- penultimate
- palindrome

Each of these functions should take exactly one parameter: the parameter must be a list. If your function was given the wrong number of parameters, we will let the interpreter complain about it. However, if the function is given the wrong type of parameter (i.e., not a list), the function should return a friendly error message (including a newline). Beginning with “USAGE: “ followed by an indication of the correct invocation. For example, “`USAGE: (palindrome {list})`”

The fifth function is

- change-head

This function takes two parameters which are both lists. If the parameters are not lists, it should return a usage message like the other functions.

Function definitions

ODDS:

`(odds lst)`
should return a new list, formed from the odd-numbered elements taken from `lst`. That is,
`(odds '(a b c d e f g))`
should return:
`(a c e g)`

`(odds (list 's 't 'u 'v 'w 'x 'y 'z))`
should return:
`(s u w y)`

`(odds '((a b) c (d e d) c (b a)))`
should return:
`((a b) (d e d) (b a))`

`(odds '())`
should return the empty list, etc.

EVENREV:

`(evenrev lst)`
should return a new list, formed from the even-numbered elements taken from `lst` but in the reverse of their original order.

That is,
`(evenrev '(a b c d e f g))`
should return:
`(f d b)`

`(evenrev (list 's 't 'u 'v 'w 'x 'y 'z))`
should return:
`(z x v t)`

`(evenrev '((h v) (j k) l (m n)))`
should return:
`((m n) (j k))`

Both `(evenrev '())` and `(evenrev '(a))`
should return the empty list, etc.

PENULTIMATE:

`(penultimate lst)`
should return a one-element list consisting of just the next-to-last item of `lst`, [or return the empty list if there were fewer than two elements in `lst`. That is,

`(penultimate '(a b c d e))`
should return:
`(d)`

`(penultimate '(a))`
should return the empty string.

PALINDROME:

`(palindrome lst)`
should return `#t` if the list `lst` is a palindrome, and return `#f` otherwise. A "palindrome" is something that reads the same backwards and forwards. That is,

`(palindrome '(s t u v w x y z))`
should return:
`#f`

`(palindrome (LIST 'j 'k 'l 'm 'l 'k 'j))`
should return:
`#t`

Both `(palindrome '())` and `(palindrome '(a))`
should return `#t`

You can receive full credit for your palindrome function as long as it works for all list that are made up entirely of atoms. If you choose to take it further, you can try to get it to behave correctly for complex lists such as:

```
(palindrome '((h i) j (k l k) j (h i)))
(palindrome '((y z) y))
```

You do not need to worry about what palindrome should do with things that are not single characters such as:

```
(palindrome '(abc))
```

CHANGE-HEAD

```
(change-head lst1 lst2)
```

returns the list generated by replacing the first element of the first list with the first element of the second list. This is a simple function; in fact, you might want to write this one first. An example:

```
(change-head '(a b c) '(d e f))
```

should return

```
(d b c)
```

```
(change-head '(a) '(b))
```

should return

```
(b)
```

```
(change-head '(a) '())
```

would result in an error

Programming in Scheme

Forgetting a closing parenthesis, your functions will not work; you will get a 'premature EOF' error. It's critical that you use an editor that will match up parentheses. In the vi editor , for example, if you position your cursor over a parenthesis and then type the '%' character, the vi cursor will jump to the matching parenthesis. If the cursor doesn't move, then you know you have a mismatch. In DrRacket, a shaded region will indicate matching parentheses.

How to set up

Dr Racket IDE: While your programs need to ultimately need to run on Linux (Gradescope, and you may practice on edoras), you may find it convenient to download DrRacket as a programming environment. https://racket-lang.org/ and select Download. Instructions to get started are on the Racket site. Open the p6.scm template file and at the top of the file add:

```
#lang scheme
```

This is necessary to run in scheme in DrRacket, *BUT YOU MUST COMMENT OUT OR REMOVE THIS STATEMENT BEFORE SUBMITTING YOUR PROGRM FOR GRADING!*

To use edoras server: At the command line prompt (in this example I'm using %), load your code (edited elsewhere) by typing:

```
% scheme --load p6.scm
```

Grading considerations:

You must provide a functional programming solution in order to get any credit. Even though the variant of Scheme on edoras allows constants to be redefined, you must NOT store values. Therefore, except for "change-head", all your four other functions must be defined recursively. See Sebesta Chapter 15 for examples.

It is okay to define "helper" functions if you find that convenient.

Your program will be checked against some predefined lists—some of these are defined in p6.scm. New lists will also be used, so think up some of your own to test your functions.

Weird caveat in the grading program: It won't attempt to do any grading unless you have *at least* defined the "odds" function into your p6.scm file. So, put a syntactically correct definition of the odds function in your p6.scm file early on (it doesn't have to work correctly, it merely has to exist and be "legal" Scheme code), or else you won't really have much of a clue as to what sort of tests the grading program will run.

In your program, be sure to put your name, have no additional input prompts, explain your algorithm, point out any failures that don't meet the requirements.

What to hand in:

Upload p6.scm to Gradescope by the due date/time. The autograder will test it and calculate a grade. Then your code will be inspected by a human grader: you can get no credit for an imperative solution and/or use variables, even if you get correct answers. You must use functional programming paradigms of recursion and functions.