

Homework 9 Solutions

hw09.zip (hw09.zip)

Solution Files

You can find the solutions in hw09.scm (hw09.scm).

Macros are a method of programming that allow programmers to treat expressions as data and create procedures of a language using the language itself. Macros open the door to many clever tricks of creating "shortcuts", and with Scheme, allow us to build our own special forms other than the ones that are built in.

Required Questions

Getting Started Videos

Getting Started Videos

These videos may provide some helpful direction for tackling the coding problems on this assignment.

To see these videos, you should be logged into your berkeley.edu email.



YouTube link (https://youtu.be/playlist?list=PLx38hZJ5RLZdiDfC9wv_3HTYtlJuEB7J)

Interpreters

Q1: WWSD: Eval and Apply

How many calls to `scheme_eval` and `scheme_apply` would it take to evaluate each of these Scheme expressions?

You may find the Interpreters Study Guide (<https://cs61a.org/study-guide/interpreters/#counting-calls>) helpful.

Use Ok to test your knowledge by writing the number of calls needed to evaluate each expression:

```
python3 ok -q wwsd-eval_apply -u
```

```
scm> (+ 2 4 6 8) ; number of calls to scheme_eval
_____
6

scm> (+ 2 4 6 8) ; number of calls to scheme_apply
_____
1

scm> (+ 2 (* 4 (- 6 8))) ; number of calls to scheme_eval
_____
10

scm> (+ 2 (* 4 (- 6 8))) ; number of calls to scheme_apply
_____
3

scm> (if #f (+ 2 3) (+ 1 2)) ; number of calls to scheme_eval
_____
5

scm> (if #f (+ 2 3) (+ 1 2)) ; number of calls to scheme_apply
_____
1

scm> (define (cube a) (* a a a)) ; number of calls to scheme_eval
_____
1

scm> (define (cube a) (* a a a)) ; number of calls to scheme_apply
_____
0

scm> (cube 3) ; number of calls to scheme_eval
_____
8

scm> (cube 3) ; number of calls to scheme_apply
_____
2
```

Toggle Solution

Macros

Q2: When Macro

Using macros, define a new special form, `when`, that has the following structure:

```
(when <condition>
  (<expr1> <expr2> <expr3> ...))
```

If the condition is not false (a truthy expression), all the subsequent operands are evaluated in order and the value of the last expression is returned. Otherwise, the entire `when` expression evaluates to `okay`.

Hint: you may find the `begin` (<https://cs61a.org/articles/scheme-spec/#begin>) form useful.

```
scm> (when (= 1 0) ((/ 1 0) 'error))
okay

scm> (when (= 1 1) ((print 6) (print 1) 'a))
6
1
a
```

```
(define-macro (when condition exprs)
  `(if ,condition ,(cons 'begin exprs) 'okay)
)
```

Use Ok to test your code:

```
python3 ok -q when-macro
```



Q3: Switch

Define the macro `switch`, which takes in an expression `expr` and a list of pairs, `cases`, where the first element of the pair is some *value* and the second element is a single expression. `switch` will evaluate the expression contained in the list of `cases` that corresponds to the value that `expr` evaluates to.

```
scm> (switch (+ 1 1) ((1 (print 'a))
                     (2 (print 'b))
                     (3 (print 'c))))
b
```

You may assume that the value `expr` evaluates to is always the first element of one of the pairs in `cases`. You can also assume that the first value of each pair in `cases` is a value.

```
(define-macro (switch expr cases)
  (cons 'cond
    (map (lambda (case) (cons `(equal? ,expr (quote ,(car case))) (cdr case)))
      cases))
)
```

Use Ok to test your code:

python3 ok -q switch



Exam Practice

Homework assignments will also contain prior exam questions for you to try. These questions have no submission component; feel free to attempt them if you'd like some practice!

Macros

1. Fall 2019 Final Q9: Macro Lens (<https://cs61a.org/exam/fa19/final/61a-fa19-final.pdf#page=10>)
2. Summer 2019 Final Q10c: Slice (<https://cs61a.org/exam/su19/final/61a-su19-final.pdf#page=10>)
3. Spring 2019 Final Q8: Macros (<https://cs61a.org/exam/sp19/final/61a-sp19-final.pdf#page=8>)

CS 61A (/)

[Weekly Schedule \(/weekly\)](#)

[Office Hours \(/office-hours\)](#)

[Staff \(/staff\)](#)

Resources (/resources)

[Studying Guide \(/articles/studying\)](#)

[Debugging Guide \(/articles/debugging\)](#)

[Composition Guide \(/articles/composition\)](#)

[Pair Programming \(/articles/pair-programming\)](#)

Policies (/articles/about)

[Assignments \(/articles/about#assignments\)](#)

[Exams \(/articles/about#exams\)](#)

[Grading \(/articles/about#grading\)](#)