CS 61A
Fall 2023

Scheme, Scheme Lists

Discussion 10: November 1, 2023

Getting Started

Someone from your group should join Discord.

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

If you have only 1 or 2 people in your group, you can join the other group in the room with you.

Pro tip: code.cs61a.org/scheme can be used to evaluate Scheme expressions in case you don't remember how something works.

Q1: Ice Breaker

Everybody say your name, and then figure out who likes Scheme the most so far. They're your group's facilitator for the day.

Facilitator: If you ever don't want to be facilitator anymore, you can ask for a new facilitator. Just say, "Would someone else be facilitator for a while?" You can even do that now.

Scheme

Brace yourselves for a midterm flashback! (Reviewing past problems is a good way to improve.)

Q2: Perfect Fit

Definition: A perfect square is `k*k` for some integer `k`.

Implement `fit`, which takes non-negative integers `total` and `n`. It returns whether there are `n` **different** positive perfect squares that sum to `total`.

Important: Don't use the Scheme interpreter to tell you whether you've implemented it correctly. Discuss! On the final exam, you won't have an interpreter.

```
;;; Return whether there are n perfect squares with no repeats that sum to total
(define (fit total n)
    (define (f total n k)
        (if (and (= n 0) (= total 0))
            #t
        (if (< total (* k k))
            #f
        'YOUR-CODE-HERE



        )))
    (f total n 1))

(expect (fit 10 2) #t)  ; 1*1 + 3*3
(expect (fit 9 1)  #t)  ; 3*3
(expect (fit 9 2)  #f)  ;
(expect (fit 9 3)  #f)  ; 1*1 + 2*2 + 2*2 doesn't count because of repeated 2*2
(expect (fit 25 1)  #t) ; 5*5
(expect (fit 25 2)  #t) ; 3*3 + 4*4
```

Use the `(or _ _)` special form to combine two recursive calls: one that uses `k*k` in the sum and one that does not. The first should subtract `k*k` from `total` and subtract 1 from `n`; the other should leaves `total` and `n` unchanged. In either case, add 1 to `k`.

Presentation Time: This version of `fit` is different from the `fit` on midterm 2, which allowed repeated perfect squares in the sum (e.g., `1*1 + 2*2 + 2*2`). As a group, come up with one sentence describing how you disallowed repeats in your implementation. Once your group agrees on an answer (or wants help), send a message to the `#discuss-queue` channel with the `@discuss` tag, your discussion group number, and the message “It fits!” and a member of the course staff will join your voice channel to hear your explanation and give feedback.

Scheme Lists & Quotation

Scheme lists are linked lists. Lightning review:

- `nil` or `()` is the empty list
- `(cons first rest)` constructs a linked list with `first` as its first element and `rest` as the rest of the list, which should always be a list.
- `(car s)` returns the first element of the list `s`.
- `(cdr s)` returns the rest of the list `s`.
- `(list ...)` takes n arguments and returns a list of length n with those arguments as elements.
- `(append ...)` takes n lists as arguments and returns a list of all of the elements of those lists.
- `(draw s)` draws the linked list structure of a list `s`. It only works on code.cs61a.org/scheme. **Try it now with something like `(draw (cons 1 nil))`**

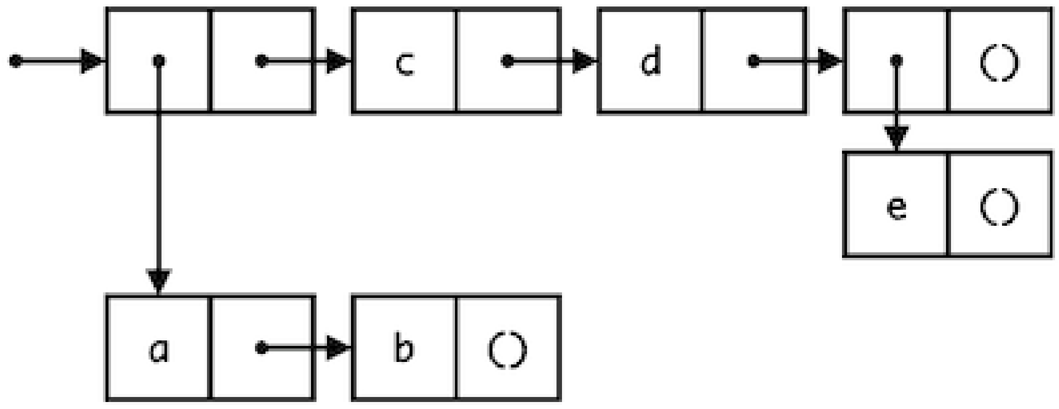
Quoting an expression leaves it unevaluated. Examples: * `'four` and `(quote four)` evaluate to the symbol `four`. * `'(2 3 4)` and `(quote (2 3 4))` evaluate to a list containing 2, 3, and 4. * `'(2 3 four)` and `(quote (2 3 four))` evaluate to a list containing 2, 3, and the symbol `four`.

Here's an important difference between `list` and quotation:

```
scm> (list 2 (+ 3 4))
(2 7)
scm> `(2 (+ 3 4))
(2 (+ 3 4))
```

Q3: Nested Lists

Create the nested list depicted below three different ways: using `list`, `quote`, and `cons`.



Facilitator: Start a conversation about how to describe the list above. Say, "It looks like there are four elements, and the first and last are lists."

First, use calls to `list` to construct this list. If you run this code in `code.cs61a.org`, the `draw` procedure will draw what you've built.

```
(define with-list
    (list
        'YOUR-CODE-HERE



    )
)
(draw with-list)
```

Every call to list creates a list, and there are three different lists in this diagram: a list containing `a` and `b`: `(list 'a 'b)`, a list containing `e`: `(list 'e)`, and the whole list of four elements: `(list _ 'c 'd _)`. Try to put these expressions together.

Now, use `quote` to construct this list.

```
(define with-quote
    '(
        'YOUR-CODE-HERE



    )

)
(draw with-quote)
```

One quoted expression is enough, but it needs to match the structure of the linked list using Scheme notation. So, your task is to figure out how this list would be displayed in Scheme.

The nested list drawn above is a four-element list with lists as its first and last elements: `((a b) c d (e))`. Quoting that expression will create the list.

Now, use `cons` to construct this list. You can use the parts provided.

```
(define helpful-list
   (cons 'a (cons 'b nil)))
(draw helpful-list)

(define another-helpful-list
    (cons 'c (cons 'd (cons (cons 'e nil) nil))))
(draw another-helpful-list)

(define with-cons
    (cons
        'YOUR-CODE-HERE



    )
)
(draw with-cons)
```

The provided `helpful-list` is the first element of the result, and `another-helpful-list` is the rest of the result, so use `cons` to put the first and rest together into a list.

Facilitator: It's time to talk about how many times `cons` appears in your last answer? Point out: "We called `cons` one time for every link in the diagram." Discuss whether that's a coincidence or not. Then, ask someone to post the number of `cons` calls in your group's Discord text chat.

Q4: Pair Up

Implement `pair-up`, which takes a list `s`. It returns a list of lists that together contain all of the elements of `s` in order. Each list in the result should have 2 elements. The last one can have up to 3.

Facilitator: Say, "Let's look at the examples together to make sure everyone understands what this procedure does."

```
;;; Return a list of pairs containing the elements of s.
;;;
;;; scm> (pair-up '(3 4 5 6 7 8))
;;; ((3 4) (5 6) (7 8))
;;; scm> (pair-up '(3 4 5 6 7 8 9))
;;; ((3 4) (5 6) (7 8 9))
(define (pair-up s)
    (if (<= (length s) 3)
        'YOUR-CODE-HERE




    ))

(expect (pair-up '(3 4 5 6 7 8)) ((3 4) (5 6) (7 8)) )
(expect (pair-up '(3 4 5 6 7 8 9)) ((3 4) (5 6) (7 8 9)) )
```

`pair-up` takes a list (of numbers) and returns a list of lists, so when `(length s)` is less than or equal to 3, return a list containing the list `s`. For example, `(pair-up (list 2 3 4))` should return `((2 3 4))`.

Use `(cons _ (pair-up _))` to create the result, where the first argument to `cons` is a list with two elements: the `(car s)` and the `(car (cdr s))`. The argument to `pair-up` is everything after the first two elements.

Discussion: What's the longest list `s` for which `(pair-up (pair-up s))` will return a list with only one element? (Don't just guess and check; discuss!) Post your answer in your group's text chat.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.