University of Virginia 25 September 2011
cs1120: Introduction of Computing
Explorations in Language, Logic, and Machines

Class 15: **Where are our Bagels?**

Upcoming Schedule

- **Monday, 3 October:** Problem Set 4
- **Wednesday, 12 October:** Exam 1 Due (will be take-home, handed out on Friday, 7 October)

Groups

Find your "bagel buddies" and work in a group on the problems.

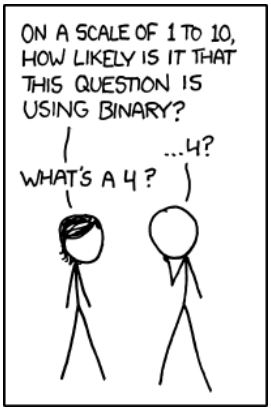
Asiago: Anthony Teate, Austin Collier, Ashita Dhir, Aleck Berry, Alex Shin
Blueberry: Anjali Nanda, Alexander Harasty, Bradley Phillips, Briana Balone, Cailey Fitzgerald
Chocolate Chip: Christopher Keshian, Casey Silver, Charity Savino, Caroline Mattey, Colton Beck
Cinnamon: Christopher Smith, Chi Zhang, Deeksha Kola, Deirdre Regan, David Crouch
Everything: Diana Naim, Emily McClure, Emmett Crawford, Filip Goc, Greald Coles
Garlic: Geoffrey Chow, Gregory Connock, Hannah Beattie, Harry Peppiatt, Irma Corado
Onion: Jiasheng Chen, Joshua Whelan, Jordan Chandler, Julia Dangtran, Jamie Miller,
Poppy Seed: Jane Willner, Kevin Liu, Kate Larsen, Li-Chang Wang, Michael Carmone
Pumpernickel: Matthew Bollinger, Margaret Neterval, Megan Dunne, Odette Kassar, Robert Jacobs
Sesame: Raven Smith, Samah Hasan, Sarah Cole, Reid Moseley, Victor Nguyen, Yuan Zhou

0. Class 14 Take-Home Problem

Design a Turing Machine that starts with an input tape that starts with a "#", is followed by a series of "∗" and "♦" symbols, followed by a "#" at the end. The output should be the number of "∗" symbols. A first version should produce the output in unary, leaving the output tape with a sequence of "1" symbols followed by a "#". For example, if the input tape is #∗♦♦∗♦∗∗♦♦∗♦♦♦# the output tape should be "#11111#". (A gold-star bonus solution would produce the output in binary notation, instead of unary.)

(If you've already worked on this problem at home as intended, discuss your approach with your groupmates who also worked on it. If not, start with the next problem.)

1. Two-Bit Comparator



Design a simple machine that computes the greater-than operator for two, two-bit inputs. It should produce a single output bit that has value **1** if the first input is greater than the second input, and value **0** otherwise.

Here are a few examples:

0 1	**0 0**	should output **1**
0 1	**0 1**	should output **0**
1 0	**0 1**	should output **1**
1 0	**1 1**	should output **0**

You can describe your machine as a Scheme expression that uses only two-input **and** and **not** procedures.

2. Building Loops

Unlike most languages, Scheme does not provide any built-in constructs for looping such as a **while** or **for** loop. For example, here is a **for** loop in Python:

```
for i in range(1, 10):
    print (i)
```

This will go through the numbers from 1 to 9 (the top of the range is not included, and execute the loop body each time with the name i having each value in turn. So, the result is printing the numbers 1 2 3 4 5 6 7 8 9 (each is printed on its own line, not as shown here).

Define a procedure that provides similar functionality in Scheme. For example,

```
(for 1 10 (lambda (i) (display i) (newline)))
```

should behave identically to the Python code above. (Hint: you may find it helpful to use a **begin** expression in your definition.) (Note: this will be useful for the last question on PS4.)

3. While Loops

The for procedure from the previous question is limited to loops where the loop index increases by one for each iteration. Define a more general while procedure, where the updating and terminating conditions are procedures. It should take four inputs: the starting (or current) value of the index, the test procedure, the update procedure, and the body procedure.

For example,

```
(while 1 (lambda (i) (< i 10)) (lambda (i) (+ i 1)) (lambda (i) (display i) (newline))))
```

would be the same as the **for** expression above. And,

```
(define (print-list p) (while p (lambda (p) (not (null? p))) cdr (lambda (p) (display (car p)) (newline))))
```

defines a procedure that prints out all the elements of the input list.