University of Virginia 27 September 2011
cs1120: Introduction of Computing
Explorations in Language, Logic, and Machines

Class 16: Making Loops

Upcoming Schedule

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

Upcoming Help Schedule

Today: 5-6:30pm (Jiamin, Rice 1st)
Thursday: 9:45-11am (Dave, Rice 507); 10-11:30am (Peter, Rice 1st); 1-2:30pm (Joseph, Rice 1st); 4:30-7:30pm (Jonathan/Jiamin, Rice 1st)

Turing Machine

Transition Rules: < *state, read symbol* > → <*next state*, *write symbol*, *direction*> | **Halt**

What does this Turing Machine do?

<*S*, 1> → <*S*, 0, *R*>
<*S*, 0> → <*S*, 1, *R*>
<*S*, #> → **Halt**

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#".

Making Loops

```
(define (for index end proc)
  (if (>= index end)
     (void)          ; this evaluates to no value
     (begin
       (proc index)
       (for (+ index 1) end proc))))
```

Use **for** to print out a multiplication table:

```
(define (while index test update proc)
  (if (test index)
      (begin
        (proc index)
        (while (update index) test update proc))
       index))
```

```
(define (loop index result test update proc)
  (if (test index)
      (loop (update index)
            (proc index result)
            test update proc)
      result))
```

```
(define (gauss-sum n)
  (loop 1 0 (lambda (i) (<= i n)) (lambda (i) (+ i 1)) _____________________________)))
```

```
(define (factorial n)
  (loop ____ _____ ____________________________
        ____________________________________________))
```

```
(define (not-null? p)  (not (null? p)))
```

```
(define (list-length p)
  (loop
```

```
(define (list-accumulate f base p)
   (loop
```