COMP 210, FALL 2000
Lecture 3: Conditional Logic

Things to do
- Check the web site for laboratory section assignments
- Reading sections 1-5

Review

Last class, we:

1. Discussed the rewriting approach to evaluating Scheme programs.
2. Saw that complex programs can be built on the shoulders of smaller programs. This led to the 1st principle of COMP 210: *Always reuse an existing function if one exists.*
3. Introduced a design methodology for programs
   a) Write a contract, purpose, and header
   b) Develop several examples, chosen to elucidate the program's behavior
   c) Write the program's body
   d) Test the program to ensure that it behaves as specified

This class, we'll apply the methodology and extend it with more detail.

In-class Exercise (3—5 minutes)
Recall our function Area

```
(define (Area Radius)
   (*  pi  (*  Radius Radius)))
```

In your New Media Design Class, you are looking at some of the tradeoffs that arise in designing a new physical format for reusable optical disks. A critical parameter is the useful area of the optical disk. We can model its area as a disk with a hole in the middle.

Design a program, named Area-of-disk-with-a-hole, that calculates the area of an optical disk. It should take two ARGUMENTs, the radius of the disk and the radius of the hole in its center. Recall the Area program that we wrote earlier.

Work with the person seated next to you. One of you will get to write your solution on the board.

The Real World

One of the hardest aspects of working for pizza is computing your net pay, as opposed to your gross pay. Recall that your gross pay could be computed as

```
; Wage: num -> num
; Purpose:  compute gross pay from hours worked
;              assume a $6 per hour pay rate
(define (Wage H)
   (*  6   H))
```

Assuming that your pay rate is $6 per hour, and that H is the number of hours worked during a week.

Under the current tax code, wages up to $500 per week are taxed at 15%, above $500 to $1,500 per week are taxed at 28%, and wages above $1,500 per week are taxed at 33%. (To keep this example simple, we assume that the 36 and 39% brackets do not exist.)

How do we write our tax program to handle this more complex tax scenario? In other words, if I give you the amount of weekly earnings, how do we figure out the taxes? This is the program "Taxes" in the previous example.

If WW stands for the weekly wage, then we first determine which tax bracket includes WW, then we multiply it by the proper tax rate.

In math, we know that we can test equality and various inequalities.

```
3 < 10          => true
10 <= 3         => false
3 = 4           => false
```

In Scheme, we can write expressions that test equalities and inequalities and return either **true** or **false**

```
(<  3  10)

(<=  10  3)

(=  3  4)
```

We can combine **true**s and **false**s using **and**, **or**, and **not**. We write **true** and **false**, in Scheme as **#t** and **#f**.

To use this insight in a program, we need to lay out the intervals on the number line, and then construct an appropriate set of tests.

```
0------------------500--------------------------------1500-------------------------
     15%                      28%                             33%
```

If WW stands for our weekly wage, then the appropriate conditions are

```
(<= WW  500)
(and  (<  500  WW)  (<=  WW  1500))
(<  1500  WW)
```

To put this together in Scheme, we use a construct called the **cond** construct (for conditional).

```
;  Real-Tax:  num -> num
; Purpose: compute the graduated income tax on a given weekly wage
(define (Real-Tax  WW)
    (cond
        ((<=  WW  500)                          (*  (/  15  100)  WW))
        ((and (<  500  WW) (<=  WW  1500))      (*  (/   28 100)  WW))
        ((<  1500 WW)                           (*   (/  33  100)  WW))  ))
```

Work an example, like (Real-Tax 300).

```
(Real-Tax 300)

=  (cond
      ((<=  300  500)                          (*  (/  15  100)  300))
      ((and (<  500  300) (<=  300  1500))     (*  (/   28 100)  300))
      ((<  1500 300)                           (*   (/  33  100)  300))  ))

=  (cond
      (#t                                      (*  (/  15  100)  300))
      ((and (<  500  300) (<=  300  1500))     (*  (/   28 100)  300))
      ((<  1500 300)                           (*   (/  33  100)  300))  ))

= (*  (/  15  100)  300)

=  45
```

When a problem specifies different behaviors for different sets of numeric data, draw an interval picture, translate the interval into conditions, and use a **cond** expression to formulate the program.

A Take-home Exercise (no pun intended)

Given all this pizza eating, we probably need to increase the amount of exercise that we do. Since all the other aspects of our pizza habit have been computerized, it seems only fair to write a program WorkOut that computes the number of hours of exercise required to counter the excess fat in pizza (as compared to that staple of undergraduate life, CK food.)

For a daily intake of	You need to work out for
0 slices	1/2 hour
1 to 3 slices	1 hour
above 3 slices	1 hour + 1/2 hour for each slice in excess of 3

Write the program (WorkOut Slices) that computes your daily exercise time, as a function of pizza slices consumed while working on COMP 210 homework.

HINT: Follow the methodology. To get started, write a contract, purpose, and header.

```
; WorkOut:  num -> num
; Purpose: compute the daily exercise period required after eating S slices of pizza
;              data relating calories to exercise supplied by the instructor
(define  (WorkOut S)
    ( … ) )
```