

Lecture 9

Fundamentals of Scheme

We will use the DrScheme version and Integrated Development Environment for this work. This is installed on all the computers in the C231 Lab. DrScheme is found under the Programming Languages Menu in the Programs choice.

Make sure that statement "Language: **Graphical Full Scheme (MrEd)**" shows in the bottom window. If it doesn't, go to the language menu and choose Change Language to Full Scheme and click on the Graphical (MrEd) button.

The upper window is called the Definitions Window. This is where you do YOUR work.

The lower window is the Interactions Window that shows results and errors.

Some talented folks at Rice University developed this environment. This is an Interpreter. That means that statements are executed directly from the source code one at a time. You can save and print your work. It has much of the same simplicity as the Turbo Pascal environment.

Note that this environment even supports a degree of windows style programming although most of your work will be output to the bottom (interactions) window. This Integrated Development Environment (IDE) is similar to the environments you are used to like CodeWarrior and Visual C++. Note also that there is a degree of "pretty printing" (formatting) supported while typing. Have fun!

Introduction to Functional Programming

Programming languages in this category have the fundamental property of being built on mathematical functions.

Mathematical functions are defined by strict rules and recursion and not by iteration.

Some examples of functions:

$$F(x) = x * x * x$$

$$G(x, y) = x + 3 * y$$

$H(x) = 1$ if $x \leq 1$ and $= x * H(x-1)$ otherwise

Mathematics uses variables but in a very different way from imperative languages like Pascal, C, and Ada. Variables in mathematics are just placeholders for values. In imperative languages, they are actual memory locations.

The Very Basics of Scheme

Scheme, developed at MIT in the mid 1970's, is a version of LISP (LISt Processing). Developers have modified Scheme to be rather powerful. DrScheme is an example of that. At some universities, Scheme is the core learning language for computer programming courses.

All programs and data in Scheme are expressions, and expressions are of two types: "atoms" and "lists." Atoms are like constants and identifiers of procedural languages like C++, Pascal, Ada, Java, etc. They are numbers, strings, names, functions and some other constructs. A list is simply a sequence of expressions separated by spaces and surrounded by parentheses. (LOTS OF PARENTHESES)

The syntax of the language is very simple:

<expression> -> <atom> | <list>
<atom> -> <number> | <string> | <identifier> | ...
<list> -> '(' <expression-sequence> ')'
<expression-sequence> -> <expression> <expression-sequence> | <expression>

Examples:

77	a number
"George"	a string
(3.04 2.3 12.23 -9.0)	a list of numbers
any	an identifier
(+ 2 3)	a list consisting of the identifier "+" and two numbers
(* (+ 2 3) (/ 6 2))	a list consisting of an identifier followed by two lists

Evaluation Rules:

- ❑ Constants evaluate to themselves
- ❑ Identifiers are looked up and replaced by the value found
- ❑ A list is evaluated by evaluating the first expression in the list. This expression must evaluate to a function. This function is then applied to the evaluated values of the rest of the list.

Examples:

(+ 10 2) Scheme looks up the function "+" and then will add 10 to the rest of the list. The value will be 12.

(+ 2 45 1 9) Scheme looks up the function "+" and then will add 2 to all of the following elements. The value will be 57.

(/ 8 2) Evaluates to 4

(/ 60 14) Evaluates to (the fraction) 30/7

(/ 80 6 2) Evaluates to (80/6/2) or 20/3

(* 5 2 3 4) Evaluates to 120

(- 90 1 2 3) Evaluates to (90 - (1+2+3)) or 84

(3 4 5) can't be evaluated because its first element is not a function

Order of Evaluations:

IN C++	IN Scheme
3 * 8	(* 3 8)
3 * 6 * 78	(* 3 (* 6 78))
gcd(45 24)	(gcd 45 24)
cin >>	(read)
cin >> value	(value (read))
cout << t	(display t)
(s < t) or (s != 1)	(or (< s t) (<> s 1))

NOTE: comments begin with ";" and end at the end of line

eg. (- 9 5) ; subtract 5 from 9

Control Statements: (Note there are NO LOOPS in Scheme. All iterative routines will be accomplished by recursion)

Two functions **if** and **cond**:

```
(if (= a 0)
    0 ;if a = 0 then return 0
    (/ 1 a) ;else return 1/a)
```

```
(cond
  ((= a 0) 0) ; if a = 0 then return 0
  ((= a 1) 1) ; else if a = 1 return 1
  (else (/ 1 a))) ; else return 1/a
```

(Note that neither the if or cond function obeys the standard evaluates rules of Scheme.)

Two functions **quote** (') and **let**:

The quote (') is often used to identify lists that **don't** start with a function like '(1 2 3).

The let function allows values to be given temporary names within an expression like

```
(let ((a 2) (b 3)) (* a b )) ;6 printed
```

((a 2) (b 3)) is called a binding list for the second expression (* a b)

Defining functions:

```
(define (quotient a)
  (if (= a 0)
      0 ;if a = 0 then return 0
      (/ 1 a))) ;else return 1/a
```

```
;f(x) = x2
(define (square x)
  (* x x))
```

```
;g(x) = 9x2 - 12.1
(define (g x)
  (- (* 9 (square x)) 12.1))
```

Although the **gcd** function (greatest common divisor) is a part of scheme, mathematically it's defined (recursively) as

$gch(x, y) = x$, if y is zero; otherwise it's equal to $gcd(y, (x \bmod y))$

```
(define (gcd x y)
  (if (= y 0)
      x
      (gcd y (remainder x y))))
(gcd 34 44) ;displays 2
```

Some Built-in Functions

max : (num num ... -> num): finds the largest number
min : (num num ... -> num): finds the smallest number
quotient : (int int -> int): integer divide
remainder : (int int -> int): arithmetic remainder
modulo : (int int -> int): the "mod" of two integers

square : (num -> num): the square of a number
sqrt : (num -> num): the square root of a number
expt : (num num -> num): the exponent of two numbers; (exp 2 5) is 2⁵
abs : (real -> real): the absolute value
exp : (num -> num): the exponential function
log : (num -> num): the natural logarithm
sin : (num -> num): the sin()
cos : (num -> num): the cos()
tan : (num -> num): the tan()
asin : (num -> num): the arcsin()
acos : (num -> num): the arccos()
atan : (num -> num): the arctan()
sinh : (num -> num): the hyperbolic sin
cosh : (num -> num): the hyperbolic cos
add1 : (number -> number): number++
sub1 : (number -> number): number--
lcm : (int int ... -> int): the least common multiple of two integers
gcd : (int int ... -> int): the greatest common divisor of two integers
numerator : (rat -> int): the numerator of a rational number; (numerator 3/5) returns 3
denominator : (rat -> int): the denominator of a rational number
random : (int -> int); a random natural number less than a given number; (random 6) will produce a random integer from 0 1 2 3 4 5