

## The Mini Lisp Interpreter

The interpreter is interactive. The user enters two kinds of inputs.

Function definitions such as

```
(define double (x) (+ x x) )
```

and expressions, such as:

```
(double 10)
```

Function definitions are simply **remembered** by the interpreter, and expressions are evaluated. Evaluating an expression is the same as **running** program in most other languages.

### Syntax

```
input--> expression | fundef
```

```
fundef --> (define function arglist expression)
```

```
arglist--> (variable*)
```

```
expression --> value | variable  
                | (if expression1 expression2 expression3 )  
                | (while expression1 expression2)  
                | (set variable expression)  
                | (begin expression+)  
                | (optr expression*)
```

```
optr --> function | value-op
```

```
value --> integer
```

value-op --> + | - | \* | / | = | < | > | **print**

function --> name

variable --> name

integer--> sequence of digits, possibly preceded by a  
minus sign

name --> any sequence of characters *not an integer*,  
and not containing (, ), ;, or **space**

A function cannot be one of the keywords **define**, **if**,  
**while**, **begin** or **set** or any of the value-ops.

Comments are introduced by the character ';' and  
continue to the end of the line.

A session is terminated by entering **quit**.

Expressions are fully parenthesized so parsing can be  
simplified. For example an expression in C

$$i = 2*j + i - k/3$$

becomes

$$(\text{set } i \text{ } (- \text{ } (+ \text{ } (* \text{ } 2 \text{ } j) \text{ } i) \text{ } (/ \text{ } k \text{ } 3)))$$

## Semantics

The meanings of expressions are presented here informally here. Note integers are the only values, so for conditional; 0 represents false and any other value represents true.

1) (**if** e1 e2 e3)

e1 evaluates to true then evaluate e2 else evaluate e3.

2) (**while** e1 e2)

Evaluate e1; if it evaluates to 0, **return 0**. Otherwise evaluate e2 and then reevaluate e1 until e1 evaluates to 0.

3) (**set** x e)

Evaluate **e** and get the value for **e**, (say v). Assign **v** to x and return **v**.

4) (**begin** e1 e2 ...en)

Evaluates each of e1, e2,...en, in that order once, and return the value of en.

5) (**f** e1 e2...en)

Evaluate each of e1,e2...en and apply that function **f** to those values. **f** may be a value-op or user defined function; if the latter its definition is found and expression defining the body is evaluated with the variables of its arglist associated with the values of e1,e2...en

**if**, **while**, **set** and **begin** are called control operators.

All value-ops take two argument except print which takes one. The arithmetic operators and the comparison operators do the obvious. **print** evaluates the argument prints it and returns the value.

Example: Greatest Common Divisor in C:

```
int gcd(int m, int n)
{
    int r = m % n;

    while ( r != 0 )
    {
        m = n;
        n = r;
        r = m % n;
    }
    return n;
}
```

To write this in mini lisp we have to define our own Operators **!=**, **%** first.

( Note we don't have ! (not) and % (mod) in our alphabet, so we have to use other characters)

```
(define not( x ) ( if x 0 1) ) ; not operator is Boolean
```

```
(define ne (x y) (not (= x y) ) )
```

```
(define mod (m n) (- m (* n (/ m n)))) )
```

```

(define gcd (m n)
  (begin
    (set r (mod m n))
    (while (ne r 0 )
      (begin
        (set m n)
        (set n r)
        (set r (mod m n))
      )
    )
    n
  )
)

```

Another recursive version:

```

(define gcd (m n)
  (if (= n 0) m (gcd n (mod m n))))

```