The Structure of Scheme Programs

A Scheme program consists of a set of function definitions. There is no structure imposed on the program and there is no main function. Function definition may be nested. A Scheme program is executed by submitting an expression for evaluation. Functions and expressions are written in the form.
(*function_name arguments*)

This syntax differs from the usual mathematical syntax in that the function name is moved inside the parentheses and the arguments are separated by spaces rather than commas. For example, the mathematical expression 3 + 4 * 5 is written in Scheme as

(+ 3 (* 4 5))

(+ 3 5)

1.Arithmetic Operators

SYMBOL & OPERATION
+ & addition
- & subtraction
* & multiplication
/ & real division

Some Examples:

(+ 4 5 2 7 5 2)
(/ 36 6 2)
(+ (* 2 2 2 2 2) (* 5 5))

2. List

Lists are the basic structured data type in Scheme. Note that in the following examples the parameters are quoted. The quote prevents Scheme from evaluating the arguments. Here are examples of some of the built in list handling functions in Scheme.

cons

takes two arguments and returns a pair (list).

(cons '1 '2) is (1 . 2)
(cons '1 '(2 3 4)) is (1 2 3 4)
(cons '(1 2 3) '(4 5 6)) is ((1 2 3) 4 5 6)

The first example is a dotted pair and the others are lists.

car

returns the first member of a list or dotted pair.

(car '(123 245 564 898)) is 123
(car '(first second third)) is first
(car '(this (is no) more difficult)) is this

cdr

returns the list without its first item, or the second member of a dotted pair.

(cdr '(7 6 5)) is (6 5)
(cdr '(it rains every day)) is (rains every day)
(cdr (cdr '(a b c d e f))) is (c d e f)
(car (cdr '(a b c d e f))) is b

null?

returns \#t if the object is the null list, ( ). It returns the null list, ( ), if the object is anything else.

list

returns a list constructed from its arguments.

(list 'a) is (a)
(list 'a 'b 'c 'd 'e 'f) is (a b c d e f)
(list '(a b c)) is ((a b c))
(list '(a b c) '(d e f) '(g h i)) is ((a b c)(d e f)(g h i))

length

returns the length of a list.

(length '(1 3 5 9 11)) is 5

reverse

returns the list reversed.

(reverse '(1 3 5 9 11)) is (11 9 5 3 1)

append

returns the concatenation of two lists.

(append '(1 3 5) '(9 11)) is (1 3 5 9 11)

3. Boolean Expression

Boolean expression returns boolean constants #T and #F.

Logical Operators

SYMBOL & OPERATION not & negation
and & logical conjunction
or & logical disjunction

Relational Operators

SYMBOL & OPERATION
= & equal (numbers)
(<) & less than
(<=) & less or equal
(>) & greater than
(>=) & greater or equal
eq? & args are identical
eqv? & args are operationally equivalent
equal? & args have same structure and contents

Example:

(equal? (car '(12 23 34)) (car '(12 23 34)))
(equal? (car '(28 23 34)) (car '(12 23 34)))
(equal?'(12 34 13) '(12 34 12))
(eq? '(12 34 13) '(12 34 12))
(eq? "abcd" "abc")

(null? '(A B))
(null? '( ))

4. Conditional Expressions

Conditional expressions are of the form:
(if *test-exp then-exp*)

(if *test-exp then-exp else-exp*).

The test-exp is a boolean expression while the then-exp and else-exp are expressions. If the value of the test-exp is true then the then-exp is returned else the else-exp is returned. Some examples include:

```
(if (> n 0) (= n 10))
(if (null? list) list (cdr list))
```

The **list** is the then-exp while **(cdr list)** is the else-exp. Scheme has an alternative conditional expression which is much like a case statement in that several test-result pairs may be listed. It takes one of two forms:

```
(cond
  (test-exp1 exp ...)
   (test-exp2 exp ...)
   ...)
(cond
  (test-exp exp ...)
   ...
  (else exp ...))
```

The following conditional expressions are equivalent.

```
(cond
      ((= n 10) (= m 1))
      ((> n 10) (= m 2) (= n (* n m)))
      ((< n 10) (= n 0)))

(cond
      ((= n 10) (=m 1))
      ((> n 10) (= m 2) (= n (* n m)))
      (else (= n 0)))
```

5. Functions

Definition expressions bind names and values and are of the form:

(define *id exp*)

Here is an example of a definition.

```
(define pi 3.14)
```

This defines pi to have the value 3.14. This is not an assignment statement since it cannot be used to rebind a name to a new value.

Lambda Expression

Nameless user defined functions are defined using lambda expressions. Lambda expressions are unnamed functions of the form:

(lambda (*id...*) *exp*)

The expression (id...) is the list of formal parameters and *exp* represents the body of the lambda expression. Here are two examples the application of lambda expressions.

((lambda (x) (* x x)) 3) is 9
((lambda (x y) (+ x y)) 3 4) is 7

User defined functions

(define (function_name parameters) (Expression))

Example:

(define (cube x) (* x x x))

Problem1: Counting number of zeros in the given list

```
(define (zeros  ls  count)
  (cond
    ((null? ls) count)
    ((zero? (car ls))
       (zeros (cdr ls) ( + count 1)))
       (else (zeros (cdr ls) count)))
       )
```

Problem 2: Write a program to reverse a list.

```
(define (reverse1 l)
(if (null? l)
     '( )
     (append (reverse1 (cdr l)) (list (car l)))
 )
)
```

Using Tail recursion:

```
(define (reverse2 list acc)
  (if (null? list)
      acc
      (reverse2  (cdr list)  (cons (car list) acc))
))
```

Problem 3: Write a program to find length of a list.

```
(define (length1 l)
(if (null? l)
     0
     (+ 1 (length1
          (cdr l)))
)
)
```

Using Tail Recursion:

```
 (define (length l acc)
(if (null? l)
     acc
     (length (cdr l) (+ acc 1))
)
)
```

Problem 4: Write a program to calculate factorial of a number.

```
(define (factorial n)
(if (<= n 1)
1
(* n (factorial (- n 1)))
)
)
```

Problem 5: Write a program to find an element is a member of the given list or not.

```
(define (member1 atm list)
  (cond
    ((null? list) #f)
    ((eq? atm (car list)) #t)
    (else (member atm (cdr list)))
    ))
```

Problem 6: Write a program to find sum of all element of list.

```
(define (sum elemList)
  (if
    (null? elemList)
    0
    (+ (car elemList) (sum (cdr elemList)))
  )
)
```

Exercise:

1. Write a program to generate the Fibonacci series.
2. Write a program to calculate GCD.
3. a.) Define a procedure in scheme “mean” that takes a non-empty list of numbers and returns the arithmetic mean of list’s elements.
   b.) Adapt the mean procedure that takes non-empty list of numbers and returns the arithmetic mean of all the elements that are numbers, ignore rest. The procedure should call error if there is no number in the list.
4. Write a program to extract a slice from list, given two indices, i and j, the slice is the list containing the elements between the i'th and j'th element of the original list (both limits included). Start counting the elements with 1.
5. If a list contains repeated elements they should be replaced with a single copy of the element. The order of the elements should not be changed.