Num class methods

```
car: a cdr: b
  ^ Num new car: a cdr: b

zero
  ^ Num car: false cdr: Nil one

decimal: n
  n = 0
    ifTrue: [^ Num car: false cdr: Nil one. ]
    ifFalse: [ ^ Num basicdecimal: n ].

basicdecimal: n
  n = 0
    ifTrue: [ ^ Nil one ]
    ifFalse: [ ^ Num car: n even not cdr: (Num basicdecimal: n // 2) ].

add: a and: b
  ^ Num add: a and: b carry: false

add: a and: b carry: c
  a nullp & b nullp
     ifTrue: [ ^ Num car: c cdr: Nil one ].
  a car
     ifTrue:
        [ b car
             ifTrue: [ ^ Num car: c cdr: (Num add: a cdr and: b cdr carry: true) ]
             ifFalse: [ ^ Num car: c not cdr: (Num add: a cdr and: b cdr carry: c) ].
  b car
     ifTrue: [ ^ Num car: c not cdr: (Num add: a cdr and: b cdr carry: c) ]
     ifFalse: [ ^ Num car: c cdr: (Num add: a cdr and: b cdr carry: false) ].

from: a subtract: b
  ^ Num from: a subtract: b borrow: false accumulate: Nil one

from: a subtract: b borrow: c accumulate: ans
  a nullp & b nullp
    ifTrue: [c
               ifTrue: [ ^ Num zero]
               ifFalse: [ ^ ans reverse ] ].
  a car
    ifTrue: [b car
               ifTrue: [ ^ Num from: a cdr
                                subtract: b cdr
                                borrow: c
                                accumulate: (Num car: false cdr: ans) ]
               ifFalse: [ ^ Num from: a cdr
                                  subtract: b cdr
                                  borrow: false
                                  accumulate: (Num car: c not cdr: ans) ] ].
  b car
    ifTrue: [ ^ Num from: a cdr
                   subtract: b cdr
                   borrow: true
                   accumulate: (Num car: c not cdr: ans) ]
    ifFalse: [ ^ Num from: a cdr
                     subtract: b cdr
                     borrow: c
                     accumulate: (Num car: c cdr: ans) ].
```

```
multiply: a by: b
  | answer |
  a zerop
    ifTrue: [ ^ a ].
  answer := (Num multiply: a half by: b) double.
  a car
    ifTrue: [ answer := Num add: answer and: b ].
  ^ answer.

divide: a0 by: b0
  | a b shifts cmp one answer bit |
  a := a0.
  b := b0.
  one := Num car: true cdr: Nil one.
  cmp := a compare: b.
  cmp = $<
    ifTrue: [ ^ Num zero ].
  cmp = $=
    ifTrue: [ ^ one ].
  shifts := Num zero.
  [ b < a ]
       whileTrue: [ b := b double.
                    shifts := shifts + one ].
  b := b half.
  answer := Nil one.
  [ shifts zerop ]
       whileFalse: [ bit := false.
                     a >= b
                        ifTrue: [ bit := true.
                                  a := a - b ].
                     answer := Num car: bit cdr: answer.
                     b := b half.
                     shifts := shifts - one ].
  ^ answer

reverse: a onto: b
  a nullp
     ifTrue: [ ^ b ]
     ifFalse: [ ^ Num reverse: a cdr onto: (Num car: a car cdr: b) ].
```

Num instance methods

```
car
  ^ hd

car: a cdr: b
  hd := a.
  tl := b

cdr
  ^ tl

double
  ^ Num car: false cdr: self

half
  ^ self cdr

zerop
```

```
  ^ self car not & self cdr zerop

nullp
  ^ false

* a
  ^ Num multiply: self by: a

+ a
  ^ Num add: self and: a

- b
  ^ Num from: self subtract: b

/ b
  ^ Num divide: self by: b

< b
  ^ (self compare: b) = $<

>= b
  ^ ((self compare: b) = $<) not

compare: n
  | answer |
  n nullp
     ifTrue: [ self zerop
                  ifTrue: [ ^ $= ]
                  ifFalse: [ ^ $> ] ].
  answer := self cdr compare: n cdr.
  answer = $=
     ifFalse: [ ^ answer ].
  self car
     ifTrue: [ n car ifFalse: [ ^ $> ] ]
     ifFalse: [n car ifTrue: [ ^ $< ] ].
  ^ $=

evaldec
  "This method is not real. Only used for printing in decimal"
  | answer |
  answer := self cdr evaldec * 2.
  self car
     ifTrue: [ ^ answer + 1 ]
     ifFalse: [ ^ answer ].

printb
  self cdr printb.
  self car
     ifTrue: [ Transcript show: 1 ]
     ifFalse: [ Transcript show: 0 ]

printd
  Transcript show: self evaldec; show: ' '

reverse
  ^ Num reverse: self onto: Nil one
```

Nil class methods

```
one
  Thenil
     ifNil: [ Thenil := Nil new ].
  ^ Thenil.
```

Nil instance methods

```
car
  ^ false

cdr
  ^ self

compare: n
  n zerop ifTrue: [ ^ $= ].
  ^ $<

double
  ^ self

evaldec
  ^ 0

half
  ^ self

nullp
  ^ true

printb
  Transcript cr

zerop
  ^ true
```