Part1

// 1)

// content in sugar2kern: [local ["A","B"] [A = false(),local ["EXU1"] [EXU1 = true(),if EXU1 then [skip/BA] else [local ["EXU2"] [EXU2 = B,if EXU2 then [skip] else [skip]]]],case A of tree() then [skip] else [case A of false() then [skip] else [case A of true() then [skip] else [skip]]]]

// for If else statements, kernel dont need to write executor to bind

// sugar2kern need to create unbound and bound executor everytime, we can just use variables directly

```
local A B in
   A = false()
   if true() then
      skip Browse A
   else
      if B then
         skip Basic
      else
         skip Basic
      end
   end
```

// content in sugar2kern: case A of tree() then [skip] else [case A of false() then [skip] else [case A of true() then [skip] else [skip]]]]

// for this one, both are almost like the same thing, dont need to write any extra executor

//

```
   case A of tree() then
      skip Basic
   end
   case A of false then
      skip Basic
    end
   case A of true then
```

```
      skip Basic
    end
end
```

```
   // 2
   // content in sugar2kern: local ["A"] [A = 2,local ["EXU1"] [local ["EXU2","EXU3"] [EXU2 = A,EXU3 =
1,"Eq" "EXU2" "EXU3" "EXU1"],if EXU1 then [skip] else [skip]],local ["EXU1"] [local ["EXU2","EXU3"]
[EXU2 = A,local ["EXU5","EXU6"] [EXU5 = 3,EXU6 = 1,"IntMinus" "EXU5" "EXU6" "EXU3"],"Eq" "EXU2"
"EXU3" "EXU1"],if EXU1 then [skip/BA] else [skip]]]
   // in sugar2kern, it need to call extra executor to bind variables or statements
   // those local variables are not allowed to use directly in it
   local A One Two Three A Res Res2 in
      One = 1
      Two = 2
      Three =3
      A = 2
      {Eq A One Res}
      if Res then
         skip Basic
      end
      {Eq A Two Res2}
      if Res then
         skip Browse A
      end
   end
```

```
   // 3
   // content in sugar2kern: local ["X","Y"] [local ["T"] [local ["EXU1","EXU2"] [EXU1 = 3,EXU2 = T,T =
tree(1:EXU1 2:EXU2)],local ["A","B","PTU0"] [PTU0 = tree(1:A 2:B),PTU0 = T,local ["EXU1"] [local
["EXU2","EXU3"] [EXU2 = 1,EXU3 = 1,"Eq" "EXU2" "EXU3" "EXU1"],if EXU1 then [local ["Z"] [local ["B"]
[local ["EXU1","EXU2"] [EXU1 = 5,EXU2 = 2,"IntMinus" "EXU1" "EXU2" "B"],skip/BB]]] else [skip]]]]]
```

```
   // this one is kind of difference with other two below, for assign a local variable, it will use a variable called "PTU0"

   // espiecially when local variables go to data record. sugar2kern will automatically assign those record to PTU0.

local T = tree(1:3 2:T) X Y in

   local tree(1:A 2:B) = T C D Five Two in

      X = 1

      Y = 1

      Five = 5

      Two = 2

      {Eq X Y C}

      {IntMinus Five Two D}

      if C then B = D Z  in

         skip Browse B

      end

   end

end


// 4

// content in sugar2kern: local ["Fun","R"] [Fun = proc {$ X EXU1} [EXU1 = X],local ["EXU1"] [EXU1 = 4,"Fun" "EXU1" "R"],skip/BR]

//for sugar2kern, number are not always the same thing.

//it used extra executor to bind the other variable, but for kelnel syntax, we dont need to use something like EXU1

// to define that, because it will automatically bind those together

// also, sugar2kern always put evey variable in side of the procedure into the definition

local Fun R in

   Fun = proc {$ X}

      R  =  X

      skip Browse R
```

```
   end
   {Fun 4}
end


// 5
// content in sugar2kern: local ["A","B"] [skip,local ["EXU1","EXU2","EXU3"] [EXU1 = 4,EXU2 = B,local ["EXU4","EXU5"] [EXU4 = B,EXU5 = B,EXU3 = '#'(1:EXU4 2:EXU5)],A = rdc(1:EXU1 2:EXU2 3:EXU3)],local ["EXU1","EXU2"] [EXU1 = 5,local ["EXU4","EXU5"] [EXU4 = 3,EXU5 = 4,"IntMinus" "EXU4" "EXU5" "EXU2"],"IntPlus" "EXU1" "EXU2" "B"],skip/BA,skip/BB,skip/s]]
// In the translation, '#' is also a type of record. instead of "B#B", in translation, it becomes " '#'(1:B 2:B)
"
local A B in
   skip Basic
   A = rdc(1:4 2:B 3:(B#B))
   local Three Four Five C in
      Three = 3
      Four = 4
      Five = 5
      {IntMinus Three Four C}
      {IntPlus C Five B}
      skip Browse A
      skip Browse B
      skip Store
   end
end


Append:
```

Append.txt

```
// Append function p 133


local Append L1 L2 L3 Out Reverse Out1 in
  Append = fun {$ Ls Ms}
             case Ls
               of nil then Ms
               [] '|'(1:X 2:Lr) then Y in
                 Y = {Append Lr Ms}
                 // skip Full
                 (X|Y)
              end
           end

  Reverse = fun {$ Ls}
              case Ls
                of nil then nil
                [] '|'(1:X 2:Xr) then Xl in
                Xl = (X|nil)
                {Append {Reverse Xr} Xl}
              end
           end



  L1 = (1|(2|(3|nil)))
  L2 = (4|(5|(6|nil)))
  L3 = (7|(8|(9|(10|nil))))
```

```
   Out = {Append L1 L2}
   Out1 = {Reverse L3}
   skip Browse Out
   skip Browse Out1
   skip Full
end

// From the information of the store, when we pass L3 into Reverse,
// it is actually doing :
// 1. {Append {Reverse [8,9,10]} [7]} ------> {Append [10,9,8] [7]}------>[10,9,8,7] (final result)
// 2. {Reverse [8,9,10]} ----> {Append {Reverse [9,10]} [8]} ------> {Append [10,9] [8]}
// 3. {Reverse [9,10]} ----> {Append {Reverse [10]} [9]} ------> {Append [10] [9]} ------>[10,9]
// 4. {Reverse [10]} ----> {Append {Reverse []} [10]} ----->[10]

// Append with difference lists

//local L1 End1 L2 End2 H1 T1 H2 T2 LNew Reverse L3 L4 Out1 in
 //  L1 = ((1|(2|End1)) # End1)       // List [1,2] as a difference list
  // L2 = ((3|(4|End2)) # End2)       // List [3,4] as a difference list
```

```
//  L1 = (H1 # T1)              // Pattern match, name head and tail
 // L2 = (H2 # T2)              // Pattern match, name head and tail
//  T1 = H2                    // Bind/unify tail of L1 with head of L2

//  LNew = (L1 # T2)            // Build a new difference list

//  skip Browse LNew

// reverse
  Reverse = fun {$ Xs}
    local ReverseD Out in
      ReverseD = fun {$ Xs Y Y1}
        case Xs
          of nil then Y = Y1 Y
          [] '|'(1:X 2:Xr) then
          {ReverseD Xr Y (X|Y1)}
        end
      end
    Out = {ReverseD Xs Out nil}
    Out
    end
  end

  L3 = (4|(3|(2|(1|nil))))

  Out1 = {Reverse L3}

  skip Browse Out1
  skip Full
```

```
end
end

// runFull "declarative" "append_diff.txt" "append_diffOut.txt"

// From the information of the store, when we pass [4,3,2,1] into Reverse
// {Reverse [4,3,2,1]} -----> {ReverseD [4,3,2,1] Out []} -----> Return [1,2,3,4]
// {ReverseD [4,3,2,1] Out []} ------> {ReverseD [3,2,1] Out [4]}
// {ReverseD [3,2,1] Out [4]} ------> {ReverseD [2,1] Out [3,4]}
// {ReverseD [2,1] Out [3,4]} ------> {ReverseD [1] Out [2,3,4]}
// {ReverseD [1] Out [2,3,4]} ------> {ReverseD [] Out [1,2,3,4]}
// {ReverseD [] Out [1,2,3,4]} ------> for the [] case, ReverseD bound [1,2,3,4] to Out, Out = [1,2,3,4]
// Return Out
```

Append environment

```
*Hoz> runFull "declarative" "append.txt" "appendOut.txt"
Out : [ 1  2  3  4  5  6 ]

Out1 : [ 10  9  8  7 ]

Store : ((106, 92), '|'(1:103 2:104)),
((105, 90, 80, 72, 33), 10),
((104, 95), '|'(1:101 2:102)),
((103, 88, 66, 31), 9),
((102, 98, 100, 97, 94, 57, 53), '|'(1:54 2:55)),
((101, 60, 29), 8),
((99, 61), nil()),
((96, 89, 85, 87, 84, 63, 59), '|'(1:60 2:61)),
((93, 91, 82), '|'(1:88 2:89)),
((86, 67), nil()),
((83, 81, 77, 79, 69, 65), '|'(1:66 2:67)),
((78, 73), nil()),
((68, 75, 71), '|'(1:72 2:73)),
((76, 34), nil()),
((74), nil()),
((70, 32), '|'(1:33 2:34)),
((64, 30), '|'(1:31 2:32)),
((62), '|'(1:80 2:81)),
((58, 28), '|'(1:29 2:30)),
((56), '|'(1:90 2:91)),
((54, 27), 7),
((55), nil()),
((52, 11), '|'(1:27 2:28)),
((51, 37), '|'(1:48 2:49)),
((50, 15), 1),
((49, 40), '|'(1:46 2:47)),
((48, 17), 2),
((47, 43, 45, 42, 39, 36, 10), '|'(1:21 2:22)),
((46, 19), 3),
((44, 20), nil()),
((41, 18), '|'(1:19 2:20)),
((38, 16), '|'(1:17 2:18)),
((35, 9), '|'(1:15 2:16)),
((25), 6),
((26), nil()),
((23), 5),
((24), '|'(1:25 2:26)),
((21), 4),
((22), '|'(1:23 2:24)),
((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["Y"] [local ["EXU2","E
```

```
((55), nil()),
((52, 11), '|'(1:27 2:28)),
((51, 37), '|'(1:48 2:49)),
((50, 15), 1),
((49, 40), '|'(1:46 2:47)),
((48, 17), 2),
((47, 43, 45, 42, 39, 36, 10), '|'(1:21 2:22)),
((46, 19), 3),
((44, 20), nil()),
((41, 18), '|'(1:19 2:20)),
((38, 16), '|'(1:17 2:18)),
((35, 9), '|'(1:15 2:16)),
((25), 6),
((26), nil()),
((23), 5),
((24), '|'(1:25 2:26)),
((21), 4),
((22), '|'(1:23 2:24)),
((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["Y"] [local ["EXU2","E
XU3"] [EXU2 = Lr,EXU3 = Ms,"Append" "EXU2" "EXU3" "Y"],local ["EXU2","EXU3"] [EXU2 = X,EXU3 = Y,EXU1 = '|'(1:EXU2 2:EXU3)]]] else [
skip]]],[("Append",8)])),
((12), '|'(1:50 2:51)),
((13), proc(["Ls","EXU1"],[case Ls of nil() then [EXU1 = nil()] else [case Ls of '|'(1:X 2:Xr) then [local ["Xl"] [local ["EXU2","E
XU3"] [EXU2 = X,EXU3 = nil(),Xl = '|'(1:EXU2 2:EXU3)],local ["EXU2","EXU3"] [local ["EXU4"] [EXU4 = Xr,"Reverse" "EXU4" "EXU2"],EXU
3 = Xl,"Append" "EXU2" "EXU3" "EXU1"]]] else [skip]]],[("Reverse",13),("Append",8)])),
((14), '|'(1:105 2:106)),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation),
((5), Primitive Operation),
((6), Primitive Operation),
((7), Primitive Operation)

Mutable Store: Empty
Current Environment : ("Append" -> 8, "L1" -> 9, "L2" -> 10, "L3" -> 11, "Out" -> 12, "Reverse" -> 13, "Out1" -> 14, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT"
-> 5, "Mod" -> 6, "IntMultiply" -> 7)
Stack : ""

*Hoz>
```

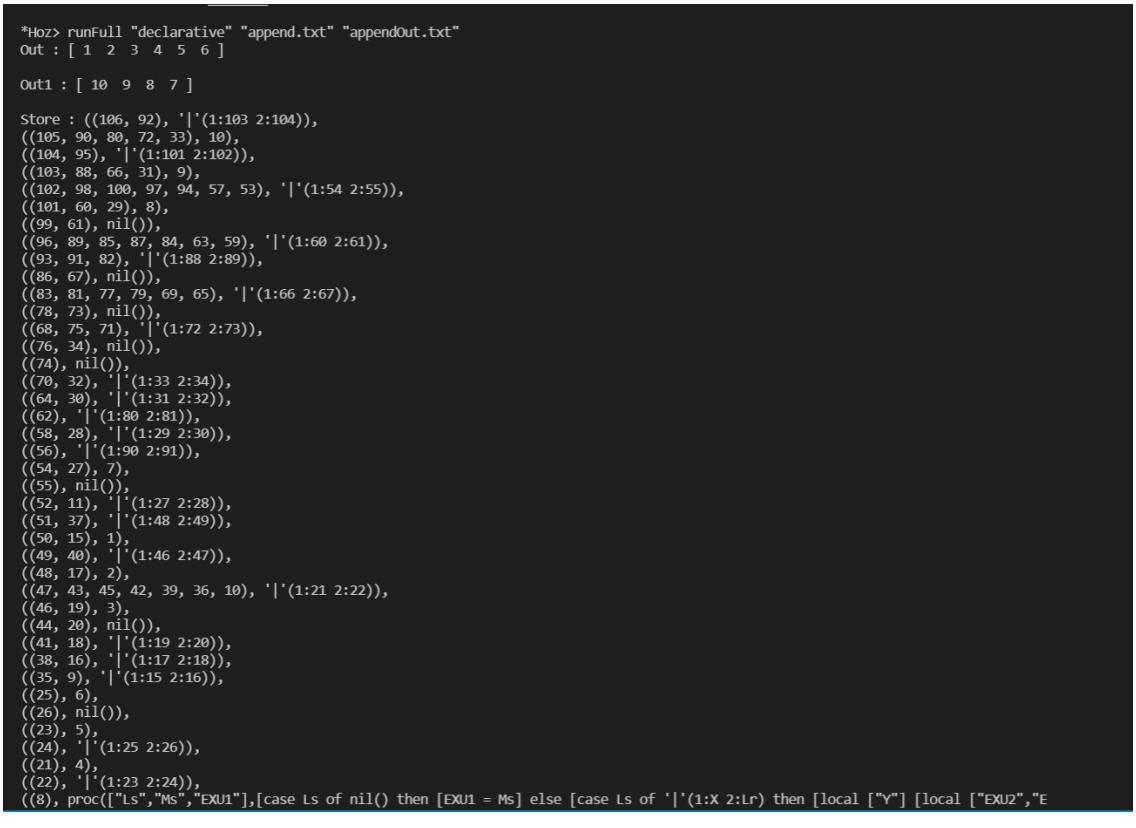
Append reverse

```
*Hoz> runFull "declarative" "append.txt" "appendOut.txt"
Out1 : [ 12  11  10  9  8  7 ]

Store : ((138, 114), '|'(1:135 2:136)),
((137, 112, 92, 77, 67, 59, 25), 12),
((136, 117), '|'(1:133 2:134)),
((135, 110, 90, 75, 53, 23), 11),
((134, 120), '|'(1:131 2:132)),
((133, 108, 88, 47, 21), 10),
((132, 123), '|'(1:129 2:130)),
((131, 106, 41, 19), 9),
((130, 126, 128, 125, 122, 119, 116, 32, 28), '|'(1:29 2:30)),
((129, 35, 17), 8),
((127, 36), nil()),
((124, 107, 103, 105, 102, 99, 96, 38, 34), '|'(1:35 2:36)),
((121, 109, 100), '|'(1:106 2:107)),
((118, 111, 97), '|'(1:108 2:109)),
((115, 113, 94), '|'(1:110 2:111)),
((104, 42), nil()),
((101, 89, 85, 87, 84, 81, 44, 40), '|'(1:41 2:42)),
((98, 91, 82), '|'(1:88 2:89)),
((95, 93, 79), '|'(1:90 2:91)),
((86, 48), nil()),
((83, 76, 72, 74, 71, 58, 46), '|'(1:47 2:48)),
((80, 78, 69), '|'(1:75 2:76)),
((73, 54), nil()),
((70, 68, 64, 66, 56, 52), '|'(1:53 2:54)),
((65, 60), nil()),
((55, 62, 58), '|'(1:59 2:60)),
((63, 26), nil()),
((61), nil()),
((57, 24), '|'(1:25 2:26)),
((51, 22), '|'(1:23 2:24)),
((49), '|'(1:67 2:68)),
((45, 20), '|'(1:21 2:22)),
((43), '|'(1:77 2:78)),
((39, 18), '|'(1:19 2:20)),
((37), '|'(1:92 2:93)),
((33, 16), '|'(1:17 2:18)),
((31), '|'(1:112 2:113)),
((29, 15), 7),
((30), nil()),
((27, 11), '|'(1:15 2:16)),
((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["Y"] [local ["EXU2","EXU3"] [EXU2 = Lr,EXU3 = Ms,"Append" "EXU2" "EXU3" "Y"],local ["EXU2","EXU3"] [EXU2 = X,EXU3 = Y,EXU1 = '|'(1:EXU2 2:EXU3)]]]
 else [skip]]],[("Append",8)])),
((9), Unbound),
((10), Unbound),
((12), Unbound),
((13), proc(["Ls","EXU1"],[case Ls of nil() then [EXU1 = nil()] else [case Ls of '|'(1:X 2:Xr) then [local ["X1"] [local ["EXU2","EXU3"] [EXU2 = X,EXU3 = nil(),X1 = '|'(1:EXU2 2:EXU3)],local ["EXU2","EXU3"] [local ["EXU4"] [EXU4 = Xr,"Reverse" "EXU4" "EXU
2"],EXU3 = X1,"Append" "EXU2" "EXU3" "EXU1"]]] else [skip]]],[("Reverse",13),("Append",8)])),
((14), '|'(1:137 2:138)),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation),
((5), Primitive Operation),
((6), Primitive Operation),
((7), Primitive Operation)

Mutable Store: Empty
Current Environment : ("Append" -> 8, "L1" -> 9, "L2" -> 10, "L3" -> 11, "Out" -> 12, "Reverse" -> 13, "Out1" -> 14, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)
Stack : ""
```

Append diff environment



```
*Hoz> runFull "declarative" "append.txt" "appendOut.txt"
Out : [ 1  2  3  4  5  6 ]

Out1 : [ 10  9  8  7 ]

Store : ((106, 92), '|'(1:103 2:104)),
((105, 90, 80, 72, 33), 10),
((104, 95), '|'(1:101 2:102)),
((103, 88, 66, 31), 9),
((102, 98, 100, 97, 94, 57, 53), '|'(1:54 2:55)),
((101, 60, 29), 8),
((99, 61), nil()),
((96, 89, 85, 87, 84, 63, 59), '|'(1:60 2:61)),
((93, 91, 82), '|'(1:88 2:89)),
((86, 67), nil()),
((83, 81, 77, 79, 69, 65), '|'(1:66 2:67)),
((78, 73), nil()),
((68, 75, 71), '|'(1:72 2:73)),
((76, 34), nil()),
((74), nil()),
((70, 32), '|'(1:33 2:34)),
((64, 30), '|'(1:31 2:32)),
((62), '|'(1:80 2:81)),
((58, 28), '|'(1:29 2:30)),
((56), '|'(1:90 2:91)),
((54, 27), 7),
((55), nil()),
((52, 11), '|'(1:27 2:28)),
((51, 37), '|'(1:48 2:49)),
((50, 15), 1),
((49, 40), '|'(1:46 2:47)),
((48, 17), 2),
((47, 43, 45, 42, 39, 36, 10), '|'(1:21 2:22)),
((46, 19), 3),
((44, 20), nil()),
((41, 18), '|'(1:19 2:20)),
((38, 16), '|'(1:17 2:18)),
((35, 9), '|'(1:15 2:16)),
((25), 6),
((26), nil()),
((23), 5),
((24), '|'(1:25 2:26)),
((21), 4),
((22), '|'(1:23 2:24)),
((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["Y"] [local ["EXU2","E
```

```
((55), nil()),
((52, 11), '|'(1:27 2:28)),
((51, 37), '|'(1:48 2:49)),
((50, 15), 1),
((49, 40), '|'(1:46 2:47)),
((48, 17), 2),
((47, 43, 45, 42, 39, 36, 10), '|'(1:21 2:22)),
((46, 19), 3),
((44, 20), nil()),
((41, 18), '|'(1:19 2:20)),
((38, 16), '|'(1:17 2:18)),
((35, 9), '|'(1:15 2:16)),
((25), 6),
((26), nil()),
((23), 5),
((24), '|'(1:25 2:26)),
((21), 4),
((22), '|'(1:23 2:24)),
((8), proc(["Ls","Ms","EXU1"],[case Ls of nil() then [EXU1 = Ms] else [case Ls of '|'(1:X 2:Lr) then [local ["Y"] [local ["EXU2","E
XU3"] [EXU2 = Lr,EXU3 = Ms,"Append" "EXU2" "EXU3" "Y"],local ["EXU2","EXU3"] [EXU2 = X,EXU3 = Y,EXU1 = '|'(1:EXU2 2:EXU3)]]] else [
skip]]],[("Append",8)])),
((12), '|'(1:50 2:51)),
((13), proc(["Ls","EXU1"],[case Ls of nil() then [EXU1 = nil()] else [case Ls of '|'(1:X 2:Xr) then [local ["Xl"] [local ["EXU2","E
XU3"] [EXU2 = X,EXU3 = nil(),Xl = '|'(1:EXU2 2:EXU3)],local ["EXU2","EXU3"] [local ["EXU4"] [EXU4 = Xr,"Reverse" "EXU4" "EXU2"],EXU
3 = Xl,"Append" "EXU2" "EXU3" "EXU1"]]] else [skip]]],[("Reverse",13),("Append",8)])),
((14), '|'(1:105 2:106)),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation),
((5), Primitive Operation),
((6), Primitive Operation),
((7), Primitive Operation)

Mutable Store: Empty
Current Environment : ("Append" -> 8, "L1" -> 9, "L2" -> 10, "L3" -> 11, "Out" -> 12, "Reverse" -> 13, "Out1" -> 14, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT"
-> 5, "Mod" -> 6, "IntMultiply" -> 7)
Stack : ""

*Hoz>
```

Append different reverse

```
*Hoz> runFull "declarative" "append_diff.txt" "append_diffOut.txt"
LNew : '#'(1:39 2:40)

Out1 : [ 12  11  10  9  8  7 ]

Store : ((20, 85, 80, 75, 70, 65, 60, 57, 55, 86), '|'(1:87 2:88)),
((88, 81), '|'(1:82 2:83)),
((87, 51), 12),
((84, 52), nil()),
((83, 76), '|'(1:77 2:78)),
((82, 49), 11),
((79, 50), '|'(1:51 2:52)),
((78, 71), '|'(1:72 2:73)),
((77, 47), 10),
((74, 48), '|'(1:49 2:50)),
((73, 66), '|'(1:67 2:68)),
((72, 45), 9),
((69, 46), '|'(1:47 2:48)),
((68, 61), '|'(1:62 2:63)),
((67, 43), 8),
((64, 44), '|'(1:45 2:46)),
((63, 58), nil()),
((62, 41), 7),
((59, 42), '|'(1:43 2:44)),
((56, 53, 18), '|'(1:41 2:42)),
((54), proc(["Xs","Y","Y1","EXU2"],[case Xs of nil() then [Y = Y1,EXU2 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU3","EXU4","EXU5"] [EXU3 = Xr,EXU4 = Y,local ["EXU6","EXU7"] [EXU6 = X,EXU7 = Y1,EXU5 = '|'(1:EXU6 2:EXU7)],"ReverseD" "EXU3" "EXU4"
 "EXU5" "EXU2"]] else [skip]]],[("ReverseD",54)])),
((40, 28, 32, 11, 37, 15), Unbound),
((39, 8, 35), '#'(1:21 2:22)),
((22, 26, 9, 34, 13, 27, 36, 14), '|'(1:29 2:30)),
((10, 38), '#'(1:27 2:28)),
((21, 33, 12), '|'(1:23 2:24)),
((31), 4),
((29), 3),
((30), '|'(1:31 2:32)),
((25), 2),
((23), 1),
((24), '|'(1:25 2:26)),
((16), '#'(1:39 2:40)),
((17), proc(["Xs","EXU1"],[local ["ReverseD","Out"] [ReverseD = proc {$ Xs Y Y1 EXU2} [case Xs of nil() then [Y = Y1,EXU2 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU3","EXU4","EXU5"] [EXU3 = Xr,EXU4 = Y,local ["EXU6","EXU7"] [EXU6 = X,EXU7 = Y1,
EXU5 = '|'(1:EXU6 2:EXU7)],"ReverseD" "EXU3" "EXU4" "EXU5" "EXU2"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Out,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4" "Out"],EXU1 = Out]],[])),
((19), Unbound),
((1), Primitive Operation),
((2), Primitive Operation),
((3), Primitive Operation),
((4), Primitive Operation),
((5), Primitive Operation),
((6), Primitive Operation),
((7), Primitive Operation)

Mutable Store: Empty
Current Environment : ("L1" -> 8, "End1" -> 9, "L2" -> 10, "End2" -> 11, "H1" -> 12, "T1" -> 13, "H2" -> 14, "T2" -> 15, "LNew" -> 16, "Reverse" -> 17, "L3" -> 18, "L4" -> 19, "Out1" -> 20, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5,
 "Mod" -> 6, "IntMultiply" -> 7)
Stack : ""
```

Part3

Reverse for append

Reverse ([1,2,3,4,5,6])

Append (reverse ([2,3,4,5,6]) ([1])

Reverse ([2,3,4,5,6]) = Append (reverse ([3,4,5,6]) [2])

Reverse ([3,4,5,6]) = Append (reverse ([4,5,6]) [3])

Reverse ([4,5,6]) = Append (reverse ([5,6]) [4])

Reverse ([5,6]) = Append (reverse ([6]) [5])

Reverse ([6]) = Append (reverse ([]) [6])

Reverse [] = []

To reverse in append, each time, they will put the smallest number in the other list

Then redo it, until both side are empty

6+5+4+3+2+1=21

Reverse for append diff

Reverse ([1,2,3,4,5,6] [])

Reverse ([2,3,4,5,6] [1])

Reverse ([3,4,5,6] [2,1])

Reverse ([4,5,6] [3,2,1])

Reverse ([5,6] [4,3,2,1])

Reverse ([6] [5,4,3,2,1])

Reverse ([] [6,5,4,3,2,1])

For this one, it move smallest to the other empty list, from the smallest to the greatest

Until left side is empty then finish

1+1+1+1+1+1+1=7