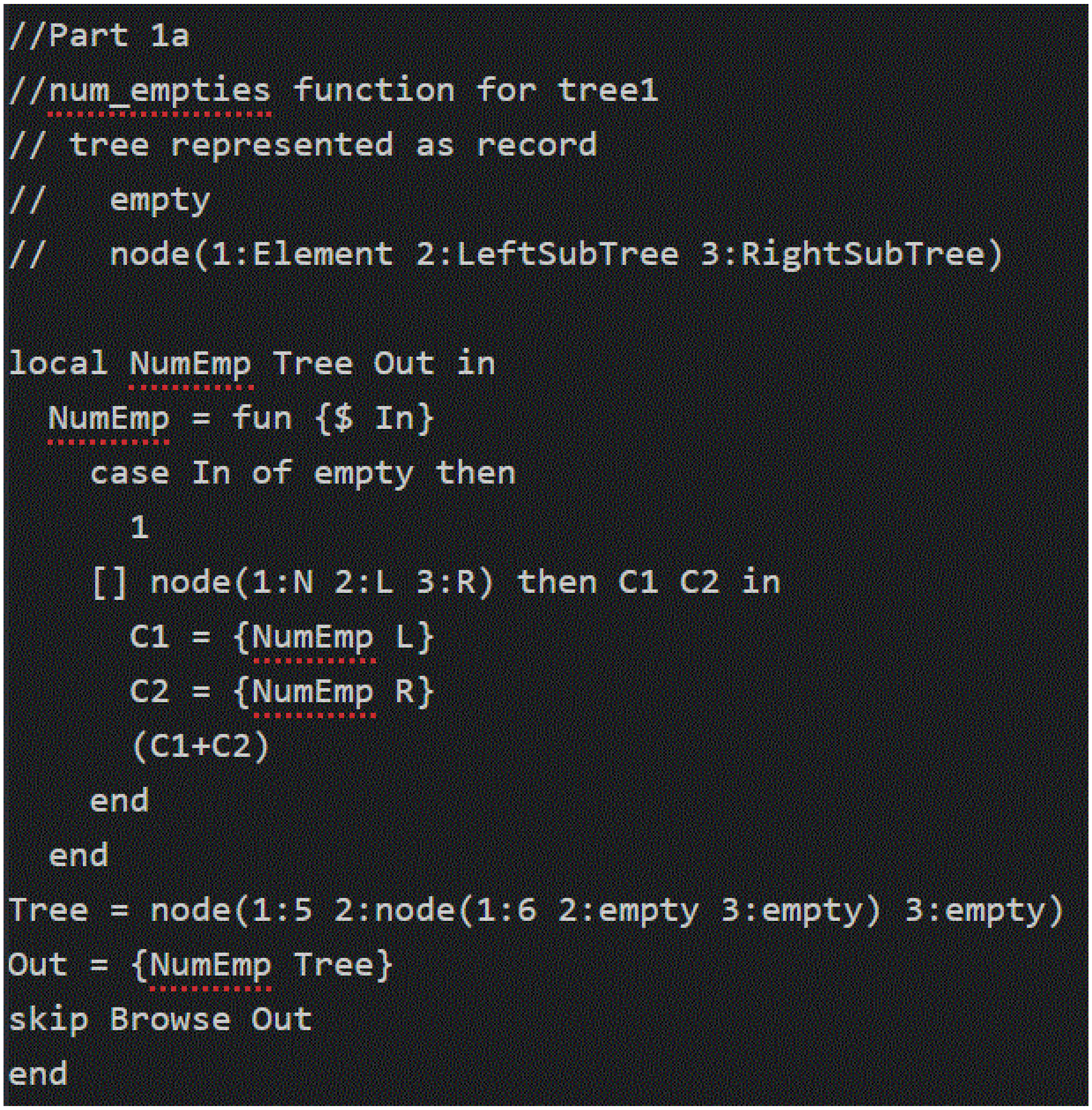
Eric Smrkovsky
9/30/2019
Csci 117
Lab6

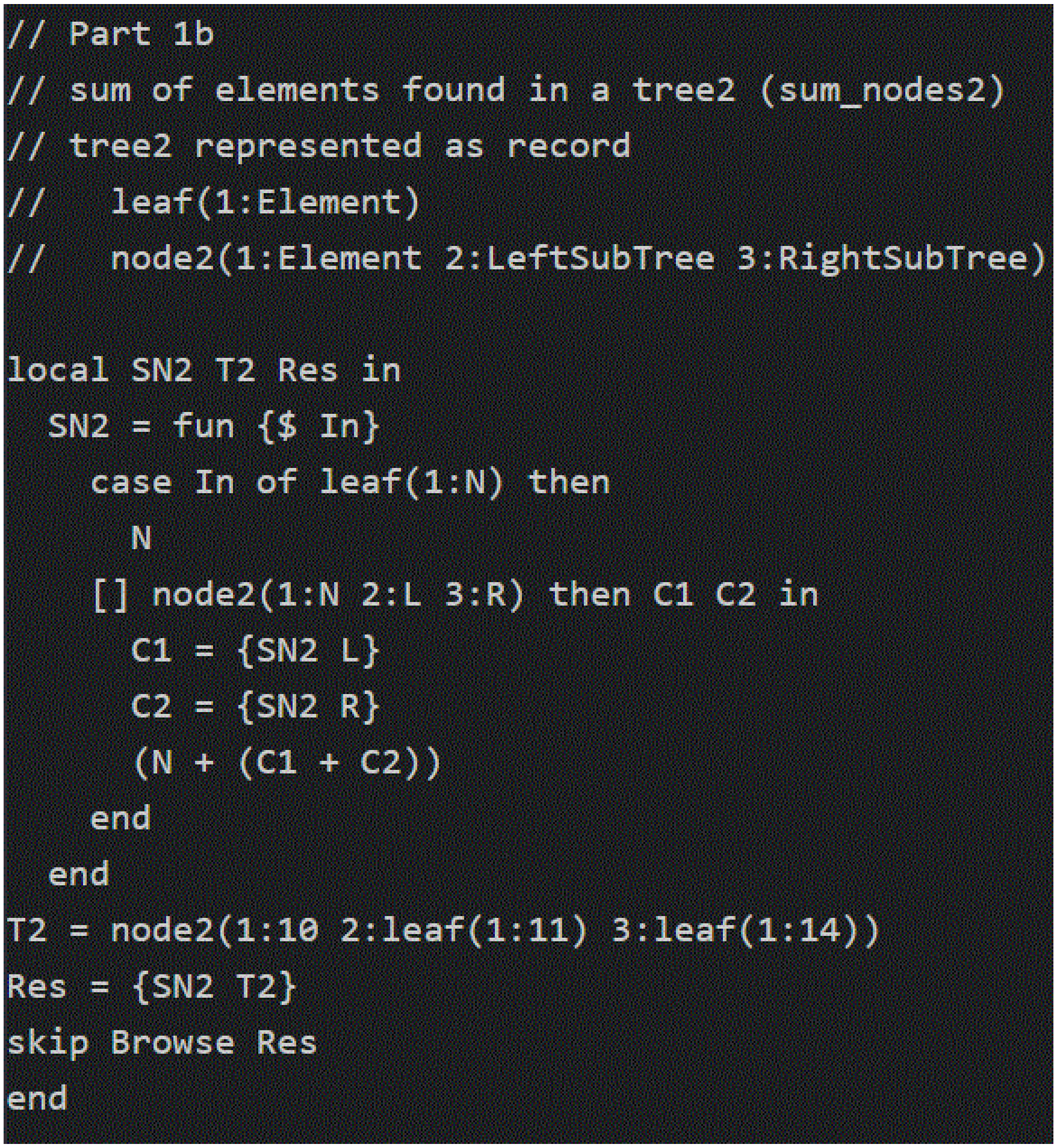
Part One



```
//Part 1a
//num_empties function for tree1
// tree represented as record
//    empty
//    node(1:Element 2:LeftSubTree 3:RightSubTree)

local NumEmp Tree Out in
  NumEmp = fun {$ In}
    case In of empty then
      1
    [] node(1:N 2:L 3:R) then C1 C2 in
      C1 = {NumEmp L}
      C2 = {NumEmp R}
      (C1+C2)
    end
  end
Tree = node(1:5 2:node(1:6 2:empty 3:empty) 3:empty)
Out = {NumEmp Tree}
skip Browse Out
end
```

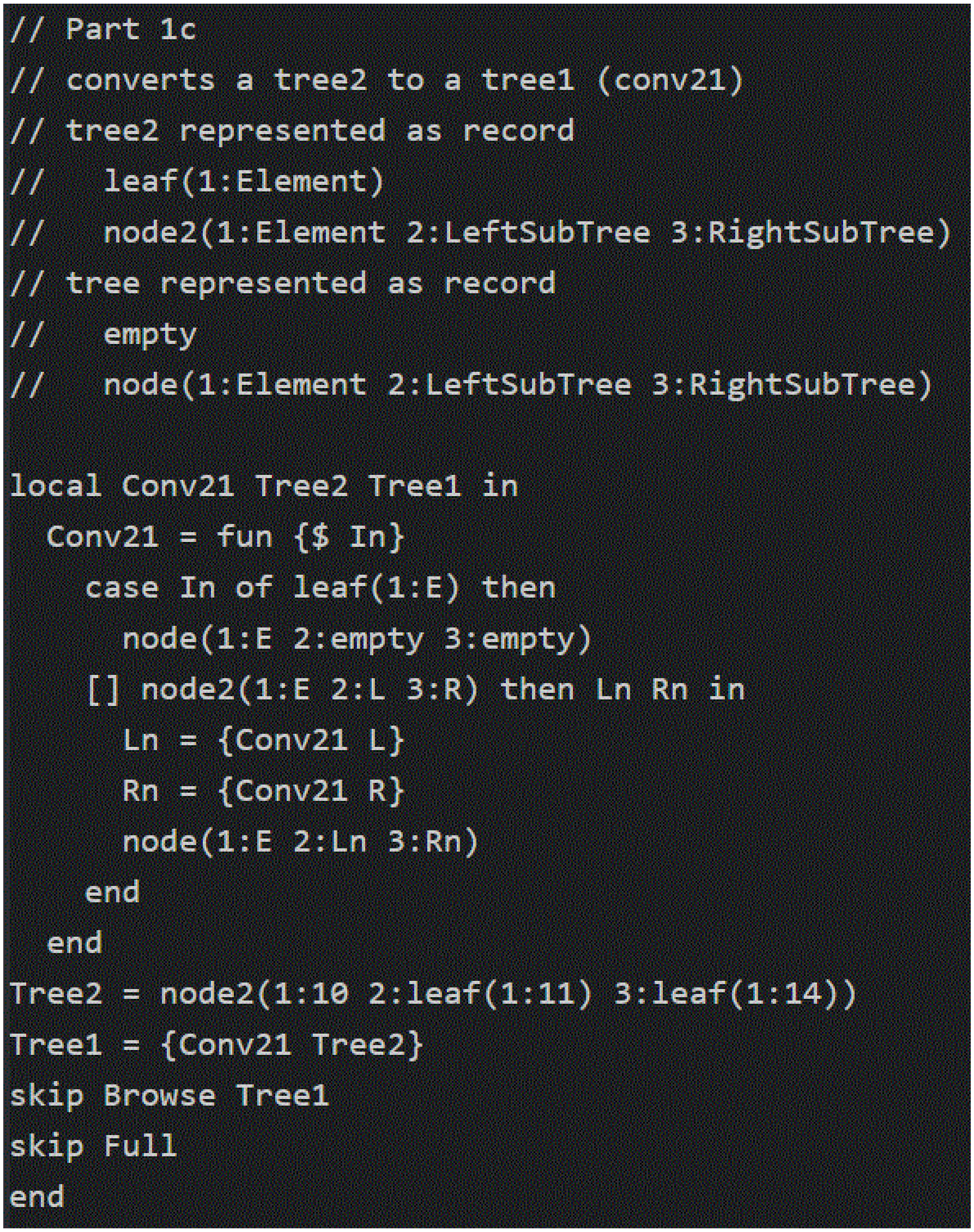
Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
// Part 1b
// sum of elements found in a tree2 (sum_nodes2)
// tree2 represented as record
//   leaf(1:Element)
//   node2(1:Element 2:LeftSubTree 3:RightSubTree)

local SN2 T2 Res in
  SN2 = fun {$ In}
    case In of leaf(1:N) then
      N
    [] node2(1:N 2:L 3:R) then C1 C2 in
      C1 = {SN2 L}
      C2 = {SN2 R}
      (N + (C1 + C2))
    end
  end
T2 = node2(1:10 2:leaf(1:11) 3:leaf(1:14))
Res = {SN2 T2}
skip Browse Res
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

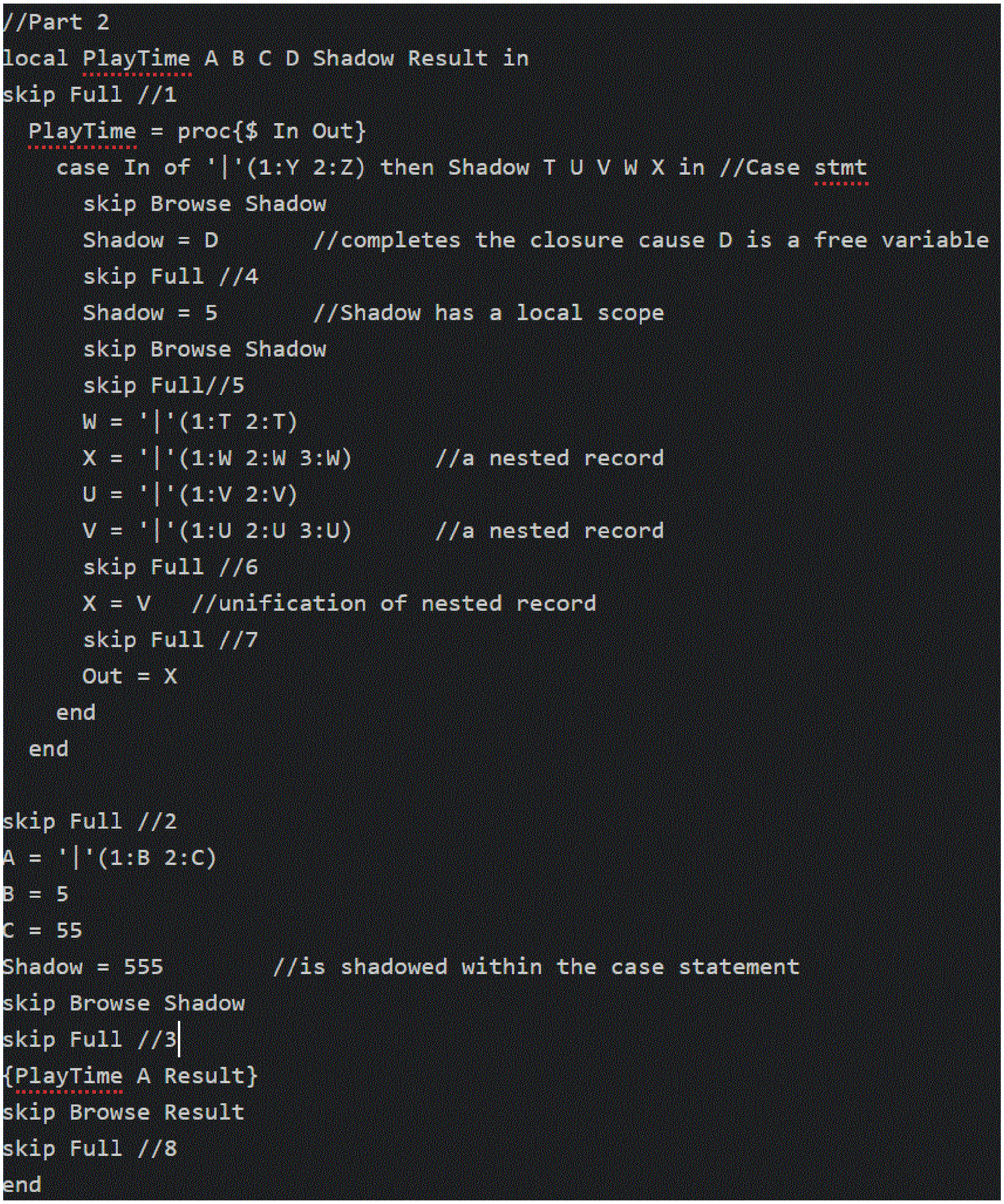


```
// Part 1c
// converts a tree2 to a tree1 (conv21)
// tree2 represented as record
//    leaf(1:Element)
//    node2(1:Element 2:LeftSubTree 3:RightSubTree)
// tree represented as record
//    empty
//    node(1:Element 2:LeftSubTree 3:RightSubTree)

local Conv21 Tree2 Tree1 in
  Conv21 = fun {$ In}
    case In of leaf(1:E) then
      node(1:E 2:empty 3:empty)
    [] node2(1:E 2:L 3:R) then Ln Rn in
      Ln = {Conv21 L}
      Rn = {Conv21 R}
      node(1:E 2:Ln 3:Rn)
    end
  end
Tree2 = node2(1:10 2:leaf(1:11) 3:leaf(1:14))
Tree1 = {Conv21 Tree2}
skip Browse Tree1
skip Full
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

Part 2



```
//Part 2
local PlayTime A B C D Shadow Result in
skip Full //1
  PlayTime = proc{$ In Out}
    case In of '|'(1:Y 2:Z) then Shadow T U V W X in //Case stmt
      skip Browse Shadow
      Shadow = D       //completes the closure cause D is a free variable
      skip Full //4
      Shadow = 5       //Shadow has a local scope
      skip Browse Shadow
      skip Full//5
      W = '|'(1:T 2:T)
      X = '|'(1:W 2:W 3:W)      //a nested record
      U = '|'(1:V 2:V)
      V = '|'(1:U 2:U 3:U)      //a nested record
      skip Full //6
      X = V   //unification of nested record
      skip Full //7
      Out = X
    end
  end

skip Full //2
A = '|'(1:B 2:C)
B = 5
C = 55
Shadow = 555        //is shadowed within the case statement
skip Browse Shadow
skip Full //3
{PlayTime A Result}
skip Browse Result
skip Full //8
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

1) At this part of the program all the variables that were just created in the local statement have been assigned store locations and in the store all of the locations remain unbound.
Environment: "Result" -> 11, "Shadow" -> 10, "D" -> 9, "C" -> 8, "B" -> 7, "A" -> 6, "PlayTime" -> 5,
Store: all unbound

2) Now a procedure has been defined so all of the work that was in that definition is removed from the stack. 5 is no longer an unbound store location it is assigned to the procedure.
Environment: same
Store: ((5), proc)

3) At this point in the program A, B, C, and Shadow have all been assigned values. The store changes with each variable's store location being bound to an integer. 6 is bound to a record which contains the store locations of the variables B and C as features. The contents of "Shadow" are displayed.
Environment: same
Store: ((10), 555), ((9), Unbound), ((8), 55), ((7), 5), ((6), '|'(1:7 2:8)),

4) Everything changes. Directly after the last time that *skip Full* was executed the procedure was called sending "A" into the function. **A case statement** is executed first within the procedure and within it's ***in*** section five new variables are introduced and assigned store locations all of which are unbound. We have the variable "Shadow" which was defined (and it's store location bound to a value) outside of the procedure being unified with the free variable "D" creating a **closure**. Also, "Shadow" is now considered a **shadowed variable** because it has two different scopes within the program, Shadow has a different store location here compared to the globally scoped Shadow, in this scope it's location is currently unbound. Shadow's store location 12 is unified with D's store location which remains unbound from the first local statement.
(new) Environment: "X" -> 17, "W" -> 16, "V" -> 15, "U" -> 14, "T" -> 13, "Shadow" -> 12, "Y" -> 7, "Z" -> 8, "In" -> 6, "Out" -> 11, "D" -> 9)
Store: ((11-17), Unbound) (9 is unified with 12)

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

5) Right away Shadow and D's store locations are bound to the integer 5 when Shadow is bound to the integer 5. Then it is displayed. Note that the value displayed is the 5 that was just bound NOT the 555 That is bound to the store location of the globally scoped Shadow. The stack contains the rest of the procedure which is six separate variable bindings. I know I haven't talked much about it, but the stack is basically just the remaining lines of code that are within whatever local scope that specific point of the program is at.
Environment: same
Store: same except, ((12, 9), 5),

6) Now the variables W X U and V have store locations that are now bound to records. Two of these records (W and U) contain variables as features whose store locations are unbound. The other two records (X and V) contain W and U as there features respectively and said features are considered **nested records**.
Environment: same
Store: ((17), '|'(1:16 2:16 3:16)), ((16), '|'(1:13 2:13)), ((15), '|'(1:14 2:14 3:14)), ((14), '|'(1:15 2:15)),

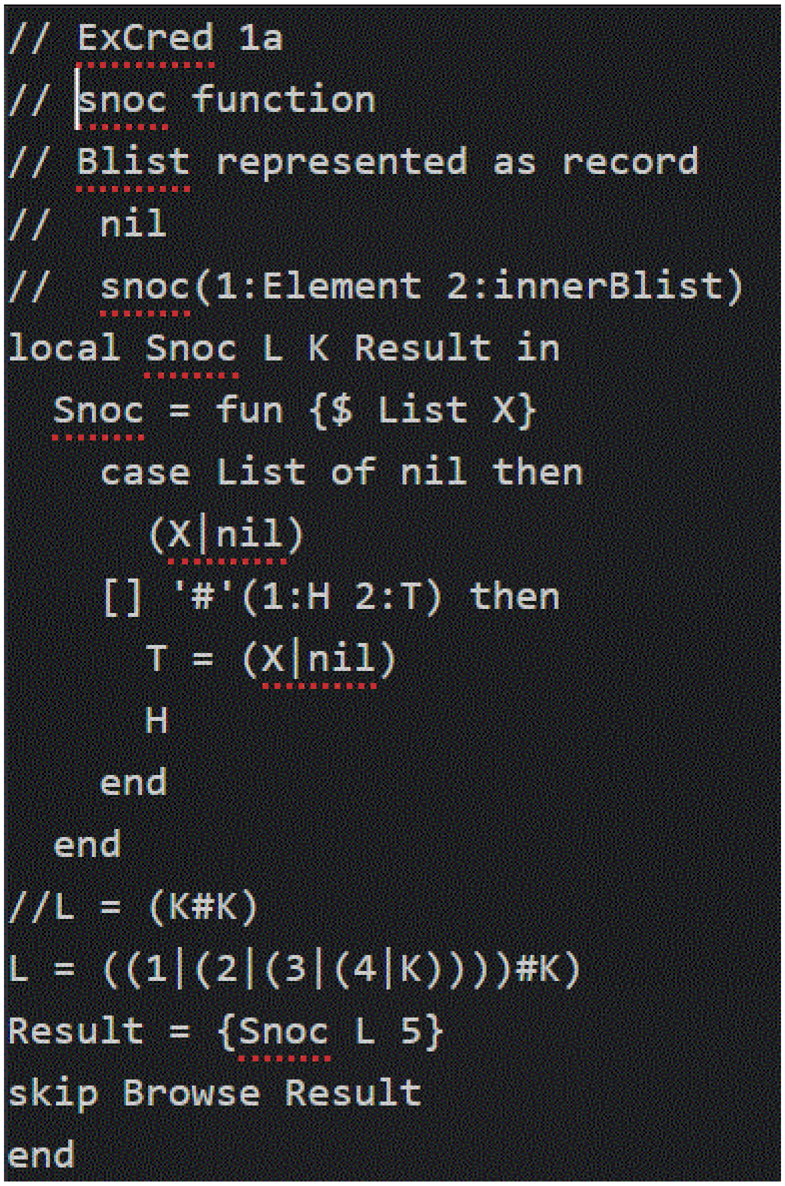
7) Now when (X=V) executes, their store locations become unified and since X and V have nested records as their features and they aren't *both already bound* they become unified as well! This is a **unification of nested records**, all of the unifications now shown in the store changes are due to the one line of code (X=V).
The last two things left in the stack are the last to remaining statements within the procedure definition.
Environment: same
Store: ((13, 17, 15), '|'(1:16 2:16 3:16)),   ((16, 14), '|'(1:13 2:13)),   ((12, 9), 5),

8) Ah, we have reached the end. But wait! One final unification has occurred! The variable group that was created when (X=V) executed is now bound to the store location that belongs to the variable Out, but when the procedure finishes the environment that contains Out isn't relevant! The relevant environment is back to the environment from the 3rd *skip Full* and the store location 11 now belongs to Result again!

Finally Result is displayed and the program terminates.

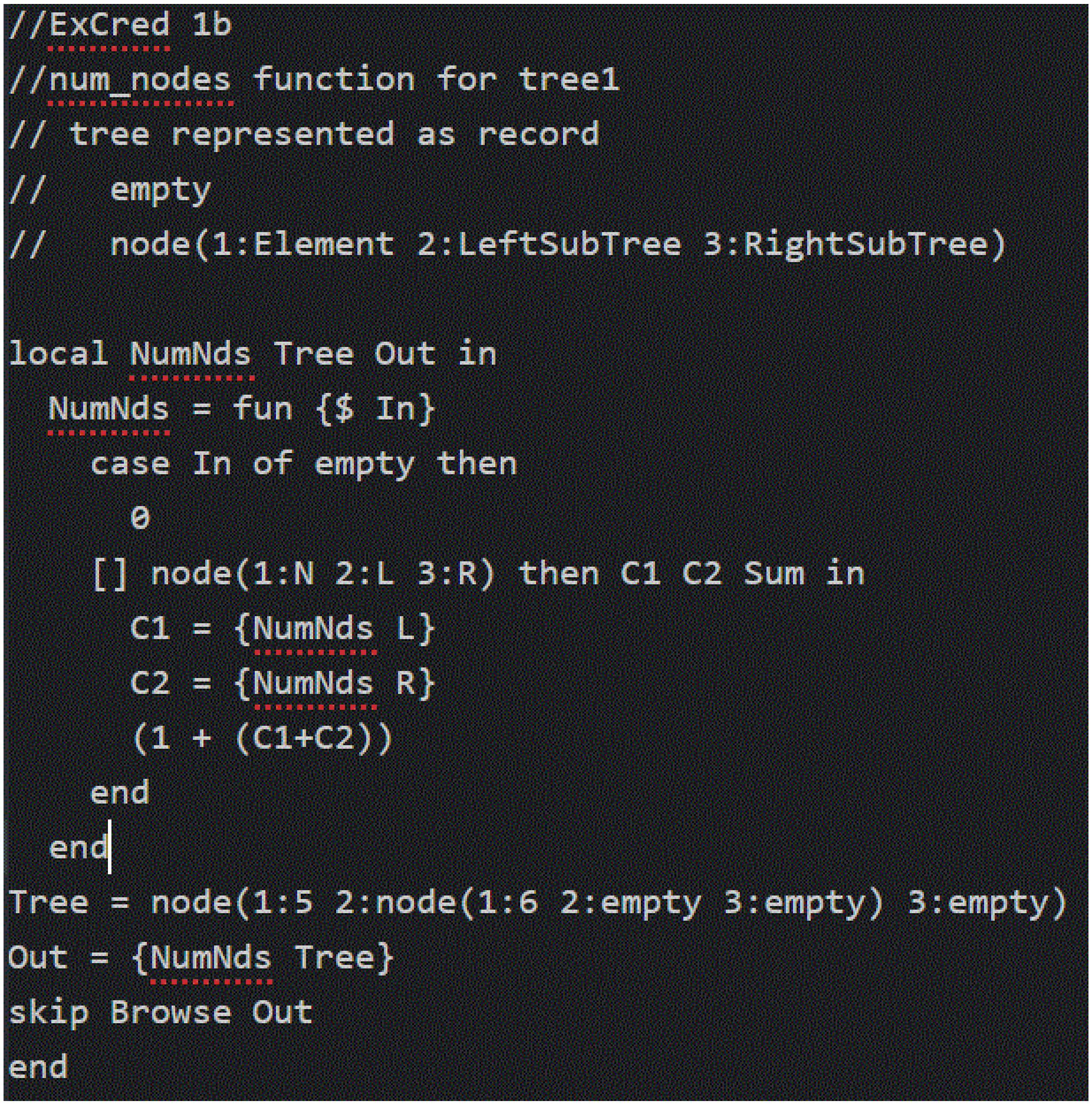
End Part2

Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
// ExCred 1a
// snoc function
// Blist represented as record
//  nil
//  snoc(1:Element 2:innerBlist)
local Snoc L K Result in
  Snoc = fun {$ List X}
    case List of nil then
      (X|nil)
    [] '#'(1:H 2:T) then
      T = (X|nil)
      H
    end
  end
//L = (K#K)
L = ((1|(2|(3|(4|K))))#K)
Result = {Snoc L 5}
skip Browse Result
end
```

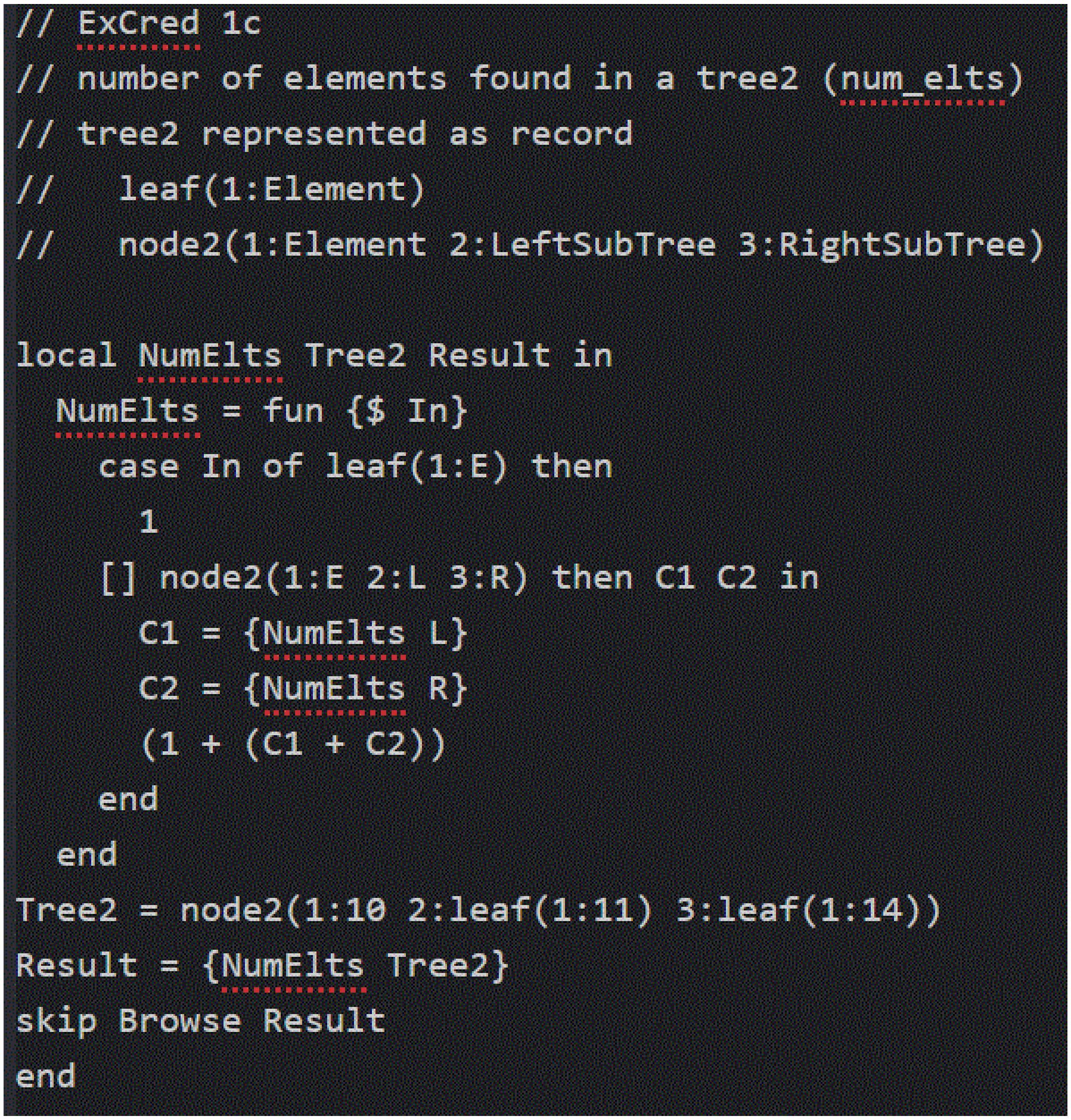
Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
//ExCred 1b
//num_nodes function for tree1
// tree represented as record
//    empty
//    node(1:Element 2:LeftSubTree 3:RightSubTree)

local NumNds Tree Out in
  NumNds = fun {$ In}
    case In of empty then
      0
    [] node(1:N 2:L 3:R) then C1 C2 Sum in
      C1 = {NumNds L}
      C2 = {NumNds R}
      (1 + (C1+C2))
    end
  end
Tree = node(1:5 2:node(1:6 2:empty 3:empty) 3:empty)
Out = {NumNds Tree}
skip Browse Out
end
```

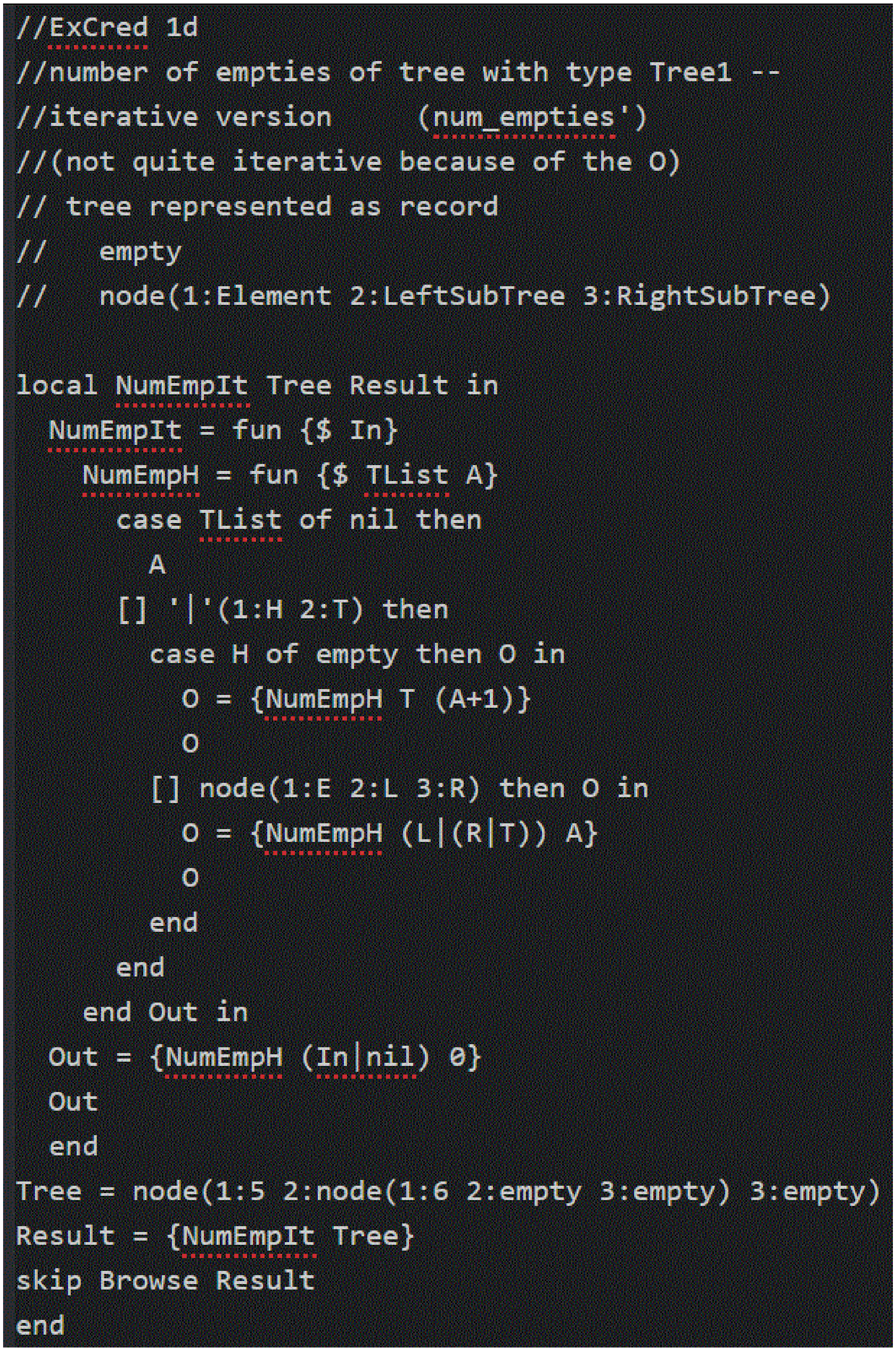
Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
// ExCred 1c
// number of elements found in a tree2 (num_elts)
// tree2 represented as record
//   leaf(1:Element)
//   node2(1:Element 2:LeftSubTree 3:RightSubTree)

local NumElts Tree2 Result in
  NumElts = fun {$ In}
    case In of leaf(1:E) then
      1
    [] node2(1:E 2:L 3:R) then C1 C2 in
      C1 = {NumElts L}
      C2 = {NumElts R}
      (1 + (C1 + C2))
    end
  end
Tree2 = node2(1:10 2:leaf(1:11) 3:leaf(1:14))
Result = {NumElts Tree2}
skip Browse Result
end
```

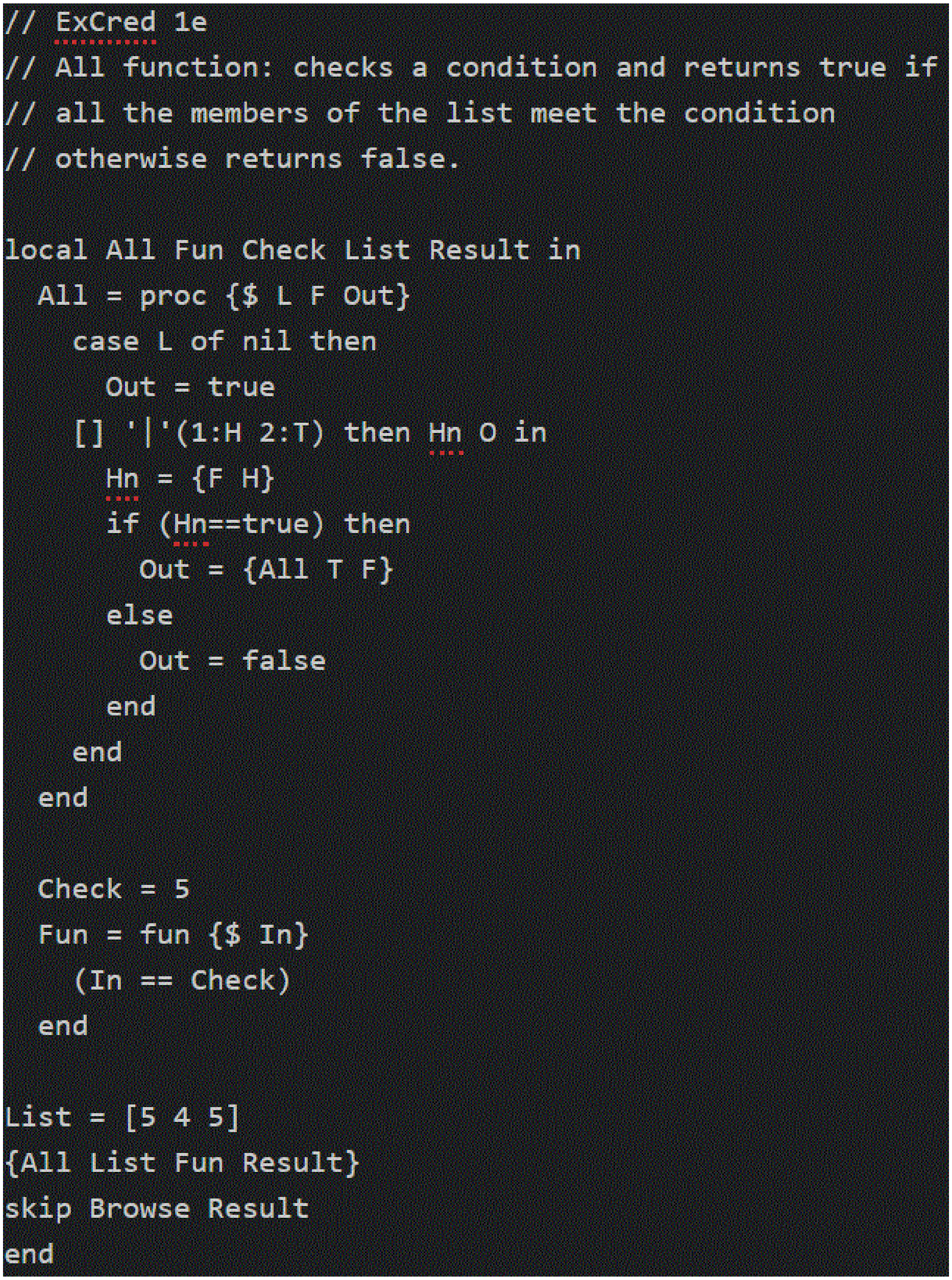
Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
//ExCred 1d
//number of empties of tree with type Tree1 --
//iterative version     (num_empties')
//(not quite iterative because of the O)
// tree represented as record
//    empty
//    node(1:Element 2:LeftSubTree 3:RightSubTree)

local NumEmpIt Tree Result in
  NumEmpIt = fun {$ In}
    NumEmpH = fun {$ TList A}
      case TList of nil then
        A
      [] '|'(1:H 2:T) then
        case H of empty then O in
          O = {NumEmpH T (A+1)}
          O
        [] node(1:E 2:L 3:R) then O in
          O = {NumEmpH (L|(R|T)) A}
          O
        end
      end
    end Out in
  Out = {NumEmpH (In|nil) 0}
  Out
  end
Tree = node(1:5 2:node(1:6 2:empty 3:empty) 3:empty)
Result = {NumEmpIt Tree}
skip Browse Result
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6



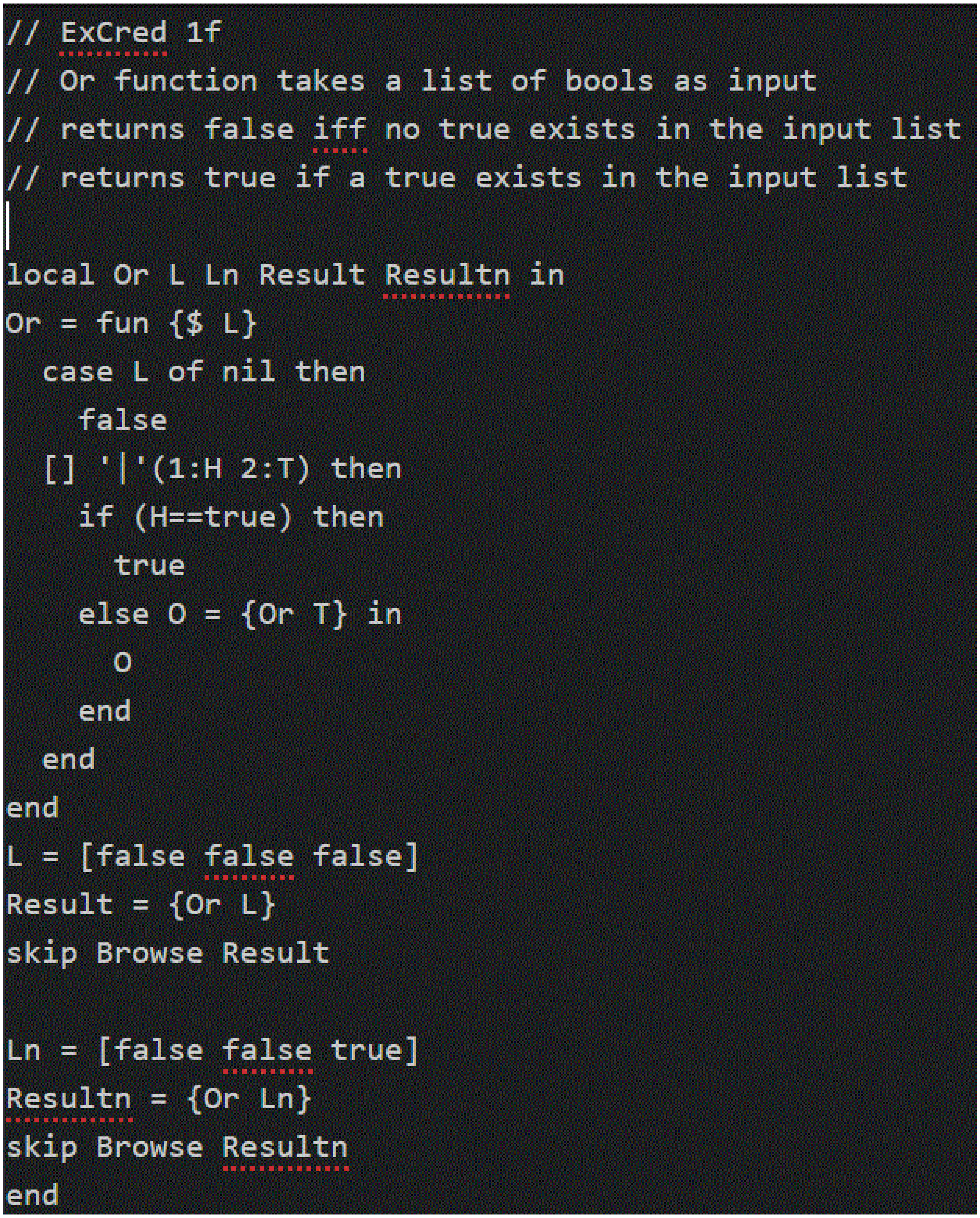
```
// ExCred 1e
// All function: checks a condition and returns true if
// all the members of the list meet the condition
// otherwise returns false.

local All Fun Check List Result in
  All = proc {$ L F Out}
    case L of nil then
      Out = true
    [] '|'(1:H 2:T) then Hn O in
      Hn = {F H}
      if (Hn==true) then
        Out = {All T F}
      else
        Out = false
      end
    end
  end

  Check = 5
  Fun = fun {$ In}
    (In == Check)
  end

List = [5 4 5]
{All List Fun Result}
skip Browse Result
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

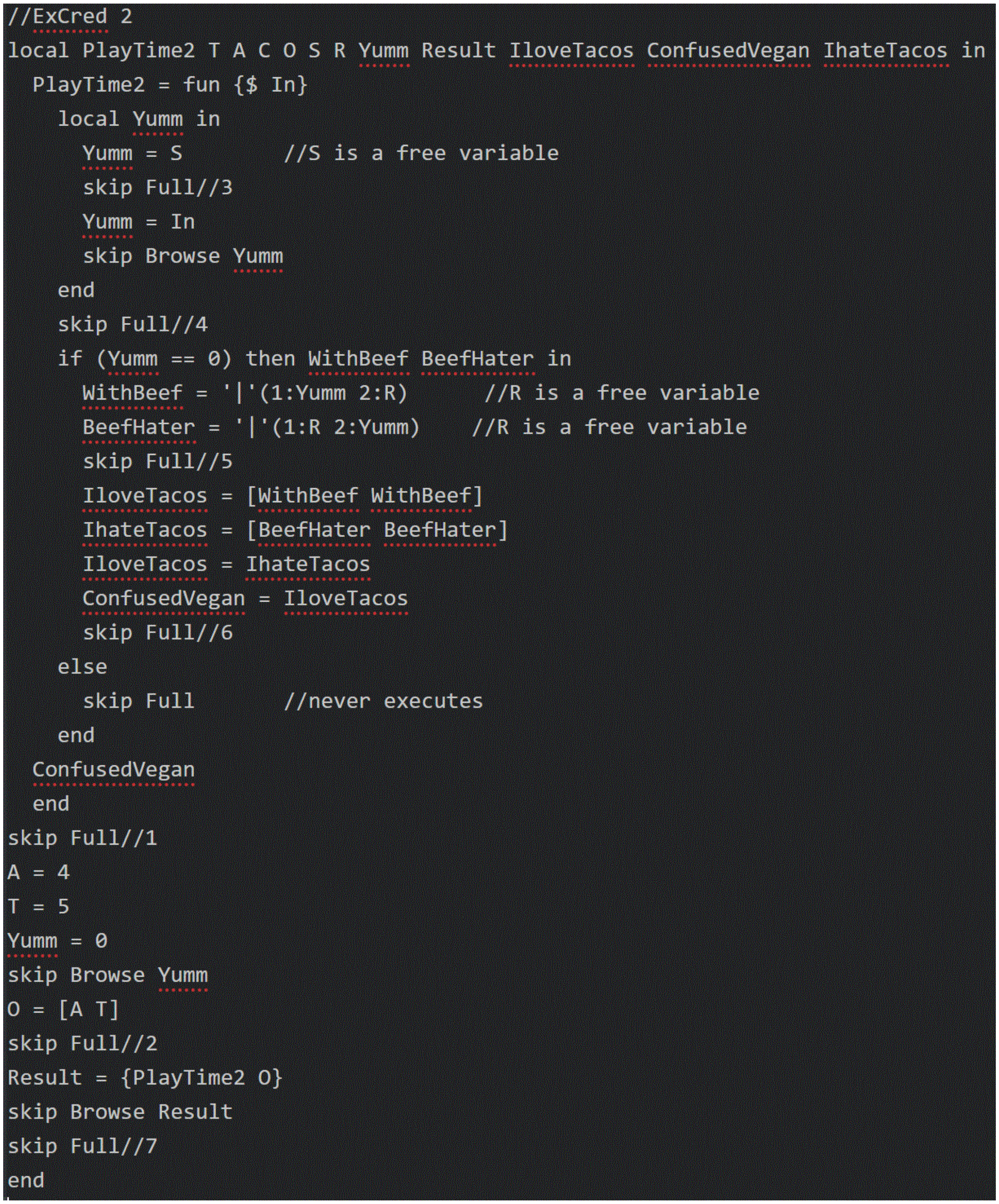


```
// ExCred 1f
// Or function takes a list of bools as input
// returns false iff no true exists in the input list
// returns true if a true exists in the input list

local Or L Ln Result Resultn in
Or = fun {$ L}
  case L of nil then
    false
  [] '|'(1:H 2:T) then
    if (H==true) then
      true
    else O = {Or T} in
      O
    end
  end
end
L = [false false false]
Result = {Or L}
skip Browse Result

Ln = [false false true]
Resultn = {Or Ln}
skip Browse Resultn
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
//ExCred 2
local PlayTime2 T A C O S R Yumm Result IloveTacos ConfusedVegan IhateTacos in
  PlayTime2 = fun {$ In}
    local Yumm in
      Yumm = S          //S is a free variable
      skip Full//3
      Yumm = In
      skip Browse Yumm
    end
    skip Full//4
    if (Yumm == 0) then WithBeef BeefHater in
      WithBeef = '|'(1:Yumm 2:R)      //R is a free variable
      BeefHater = '|'(1:R 2:Yumm)    //R is a free variable
      skip Full//5
      IloveTacos = [WithBeef WithBeef]
      IhateTacos = [BeefHater BeefHater]
      IloveTacos = IhateTacos
      ConfusedVegan = IloveTacos
      skip Full//6
    else
      skip Full       //never executes
    end
  ConfusedVegan
  end
skip Full//1
A = 4
T = 5
Yumm = 0
skip Browse Yumm
O = [A T]
skip Full//2
Result = {PlayTime2 O}
skip Browse Result
skip Full//7
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

Extra Credit 2 Description

1. At this point in the program 12 variables have been created by the local statement, all of them are unbound except for the variable PlayTime2 whose store location has been assigned to a function definition. In the Environment each variable has been assigned a store location.

Environment: ("IhateTacos" -> 16, "ConfusedVegan" -> 15, "IloveTacos" -> 14, "Result" -> 13, "Yumm" -> 12, "R" -> 11, "S" -> 10, "O" -> 9, "C" -> 8, "A" -> 7, "T" -> 6, "PlayTime2" -> 5, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4)

Store: ((16), Unbound),((15), Unbound),((14), Unbound),((13), Unbound),((12), Unbound),((11), Unbound),((10), Unbound),((9), Unbound),((8), Unbound),((7), Unbound),((6), Unbound),((5), proc),

2. At this point in the program the store locations that belong to the variables A T and Yumm have all been assigned integer values. The value within the store location belonging to the variable Yumm is displayed with a skip statement. Next the variable O's store location is bound to a record with A and T as it's features

Environment: same

Store: ((18), nil()),((17), '|'(1:6 2:18)), ((12), 0), ((9), '|'(1:7 2:17)), ((7), 4),((6), 5),

3. This next point in the program has had three things happen. Firstly, the procedure was called passing O as it's input. Fist in the procedure is a local statement that adds a locally scoped and **shadowed variable** Yum , then assigns it the same unbound store location that belongs to the **free variable** S. This also creates a **non-empty closure.** in is also assigned a store location so that it can access what was passed into the function. A whole new environment containing some new variables with with new store locations. The reason why the last three are included is because the function has already checked to make sure it has an output. Within that output is a variable group that all have the same record as it's value.

(new) Environment: ("Yumm" -> 20, "In" -> 19, "ProcOut0" -> 13, "S" -> 10, "Yumm" -> 12, "Eq" -> 3, "R" -> 11, "IloveTacos" -> 14, "IhateTacos" -> 16, "ConfusedVegan" -> 15)

Store: ((20, 10), Unbound),((19, 9), '|'(1:7 2:17)),

Eric Smrkovsky
9/30/2019
Csci 117
Lab6

4. Now we have just left the scope of the previous local statement. Yumm is still bound to the same store location as the free variable S, and both are now assigned what was passed as input to the function (O). We use a skip statement to show the user that Yumm is not the same as when it was shown from the outer scope of the program. All of the Variable that was produced with the local statement are now not a part of the environment anymore.

Environment: same except Yumm with location 20 is gone.

Store: ((20, 10, 19, 9), '|'(1:7 2:17)), (In Yumm S and O all have store locations in same variable group)

5. Next an **If statement** has started to execute. The contents of the store location that Yumm (14) are checked for equality as the condition of the if statement, returning true and the boolean is added to the If's store. During the check a new store location for the value zero is created and added to that values variable group. The variables WithBeef and BeefHater are assigned store locations and the locations populated with records. One record has its second feature unbound, and the second record has it's first feature unbound. These are unbound because the variable in those positions have a store location that is unbound.

Environment: "BeefHater" -> 25, "WithBeef" -> 24, "UniqueIF" -> 21,

Store: ((25), '|'(1:11 2:12)),((24), '|'(1:12 2:11)),((22, 12), 0),((23), 0),((21), true()),

6. At this point in the program we have reached the end of an **if** statement. When lines 15 and 16 execute two **nested records** are produced. The very next line sets them equal. When this happens, the parser has some work to do. It is necessary to check to see if it is possible to bind these two variables together. Either their arity must be the same or unbound for this to work. In the previous skip Full description, I stated how the WithBeef and BeefHater each had an unbound variable in their features, well here they are the members of the two variables that are attempting to bind. Since the unbound features match well with the bound features, it is successful. After this bind, one more bind is done, binding the variables with store locations of 14, 15 and 16 together into one variable group. Since the features of these variables were able to match, they are placed in their own variable group, as well as their features too.

Environment: same

Store: ((15, 14, 16), '|'(1:24 2:26)),((27, 29), nil()),((26, 28), '|'(1:24 2:27)),((22, 12, 11), 0),

((24, 25), '|'(1:12 2:11)),

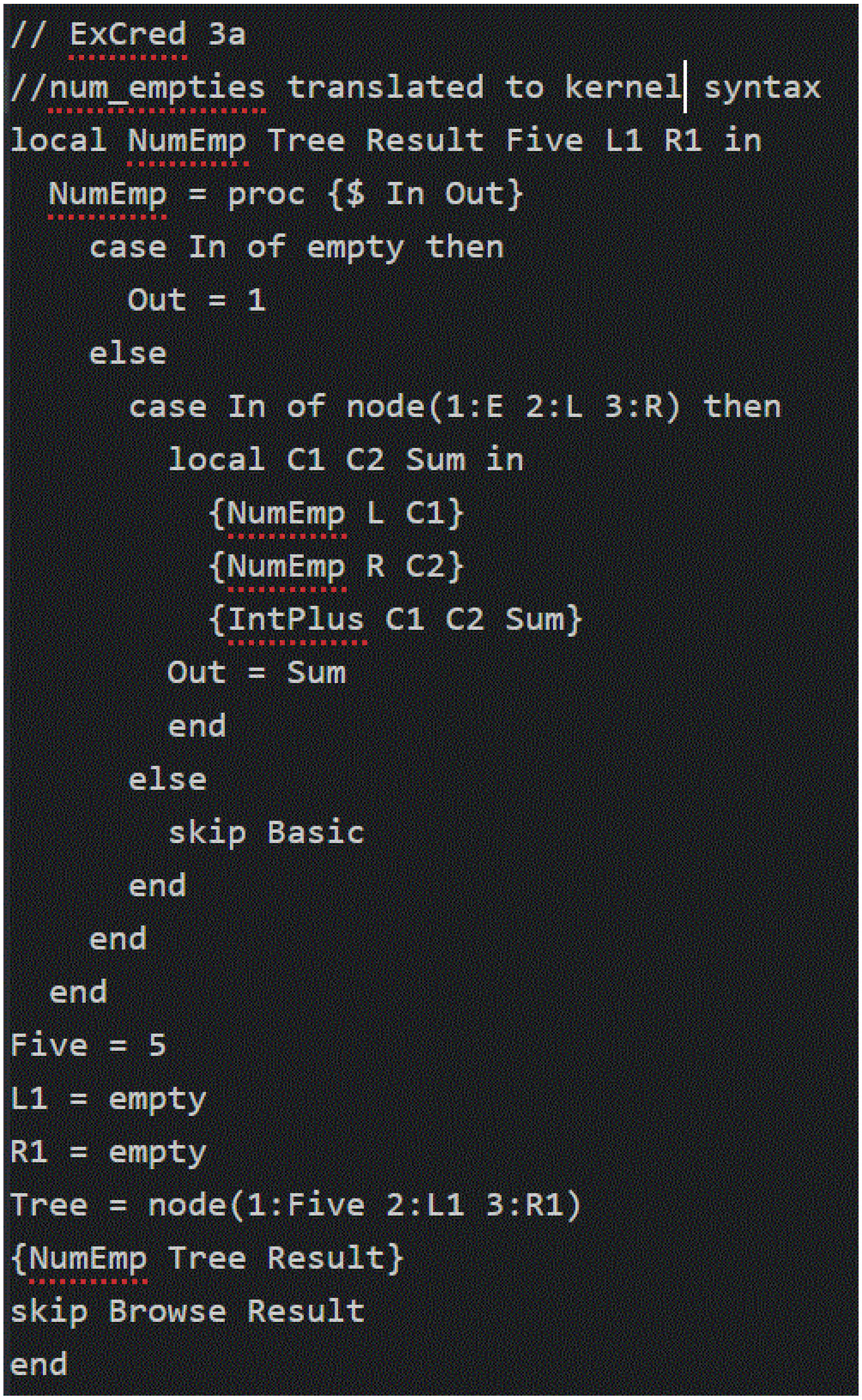
Eric Smrkovsky
9/30/2019
Csci 117
Lab6

7. Well, we have reached the end of the program. After skipping over an else statement, the function finishes, returning the store location belonging to the variable ConfusedVegan. Outside of the function, the variable Result's store location is added to the variable group belonging to the store locations of the variables ConfusedVegan, BeefHater and WithBeef. The contents of the store location belonging to the variable Result is displayed with a final skip Browse call and it does not look like it is edible. Time for lunch.

Environment: Same as part one of this description

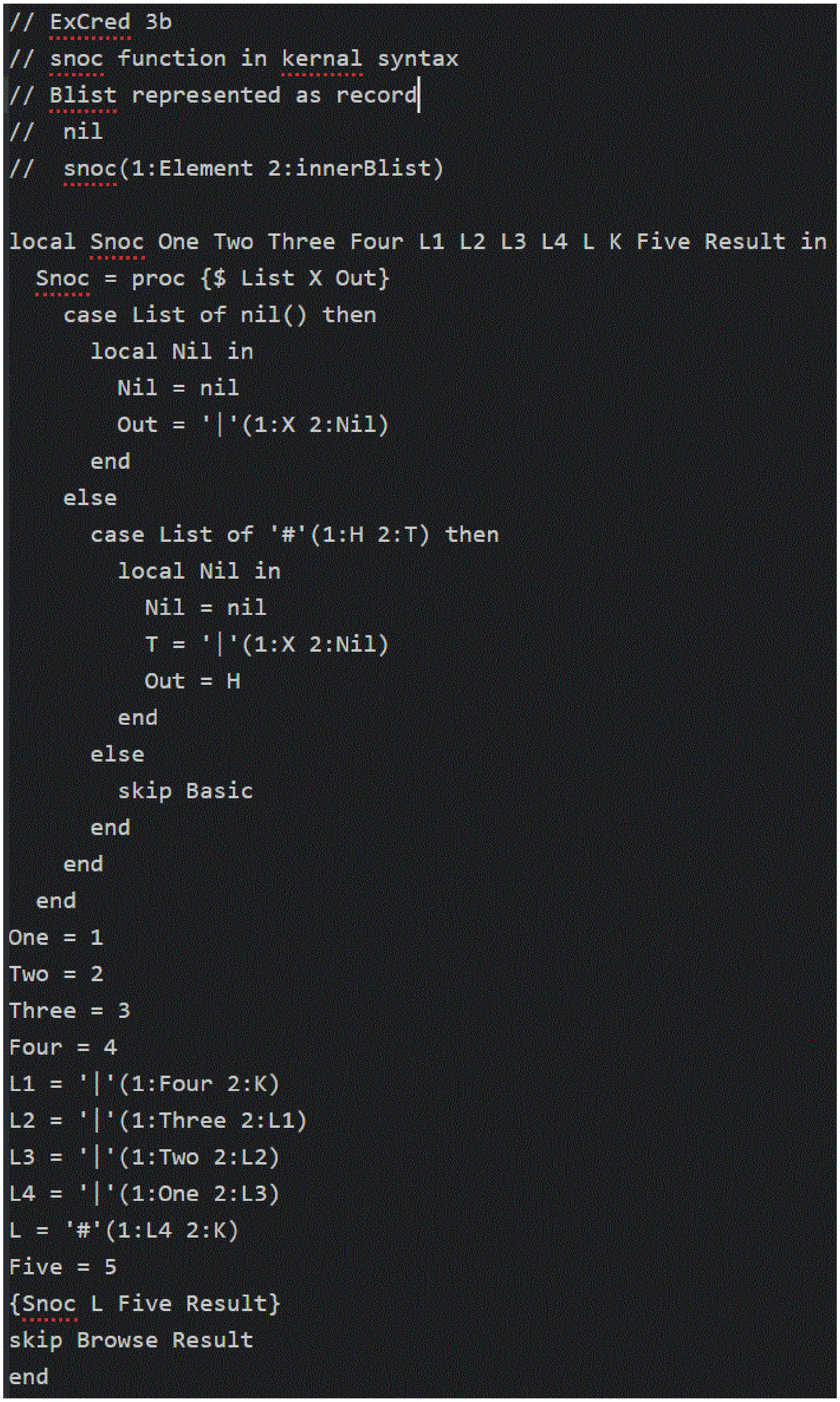
Store: ((13, 15, 14, 16), '|'(1:24 2:26)),

Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
// ExCred 3a
//num_empties translated to kernel syntax
local NumEmp Tree Result Five L1 R1 in
  NumEmp = proc {$ In Out}
    case In of empty then
      Out = 1
    else
      case In of node(1:E 2:L 3:R) then
        local C1 C2 Sum in
          {NumEmp L C1}
          {NumEmp R C2}
          {IntPlus C1 C2 Sum}
        Out = Sum
        end
      else
        skip Basic
      end
    end
  end
Five = 5
L1 = empty
R1 = empty
Tree = node(1:Five 2:L1 3:R1)
{NumEmp Tree Result}
skip Browse Result
end
```

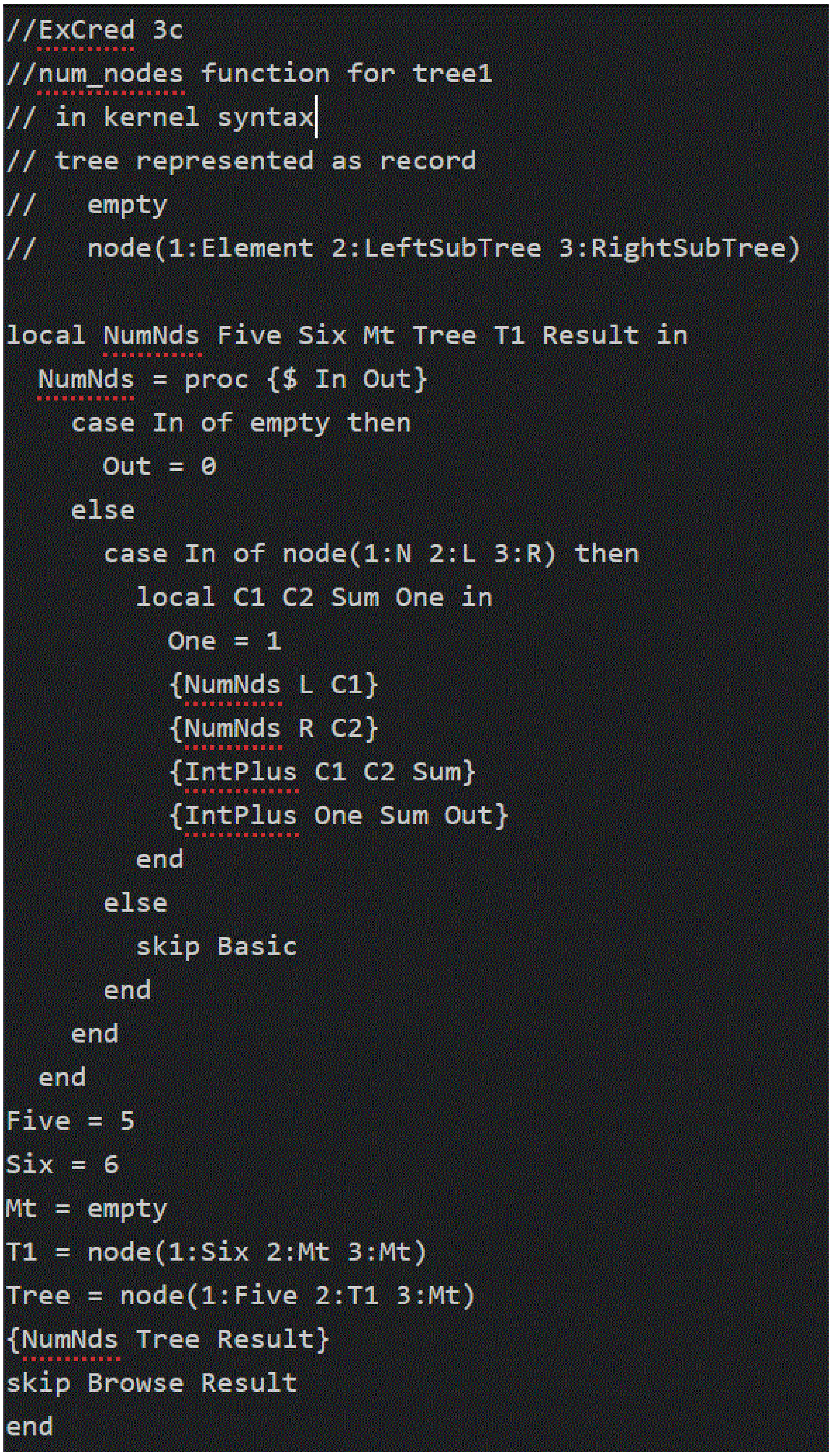
Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
// ExCred 3b
// snoc function in kernal syntax
// Blist represented as record
//  nil
//  snoc(1:Element 2:innerBlist)

local Snoc One Two Three Four L1 L2 L3 L4 L K Five Result in
  Snoc = proc {$ List X Out}
    case List of nil() then
      local Nil in
        Nil = nil
        Out = '|'(1:X 2:Nil)
      end
    else
      case List of '#'(1:H 2:T) then
        local Nil in
          Nil = nil
          T = '|'(1:X 2:Nil)
          Out = H
        end
      else
        skip Basic
      end
    end
  end
One = 1
Two = 2
Three = 3
Four = 4
L1 = '|'(1:Four 2:K)
L2 = '|'(1:Three 2:L1)
L3 = '|'(1:Two 2:L2)
L4 = '|'(1:One 2:L3)
L = '#'(1:L4 2:K)
Five = 5
{Snoc L Five Result}
skip Browse Result
end
```

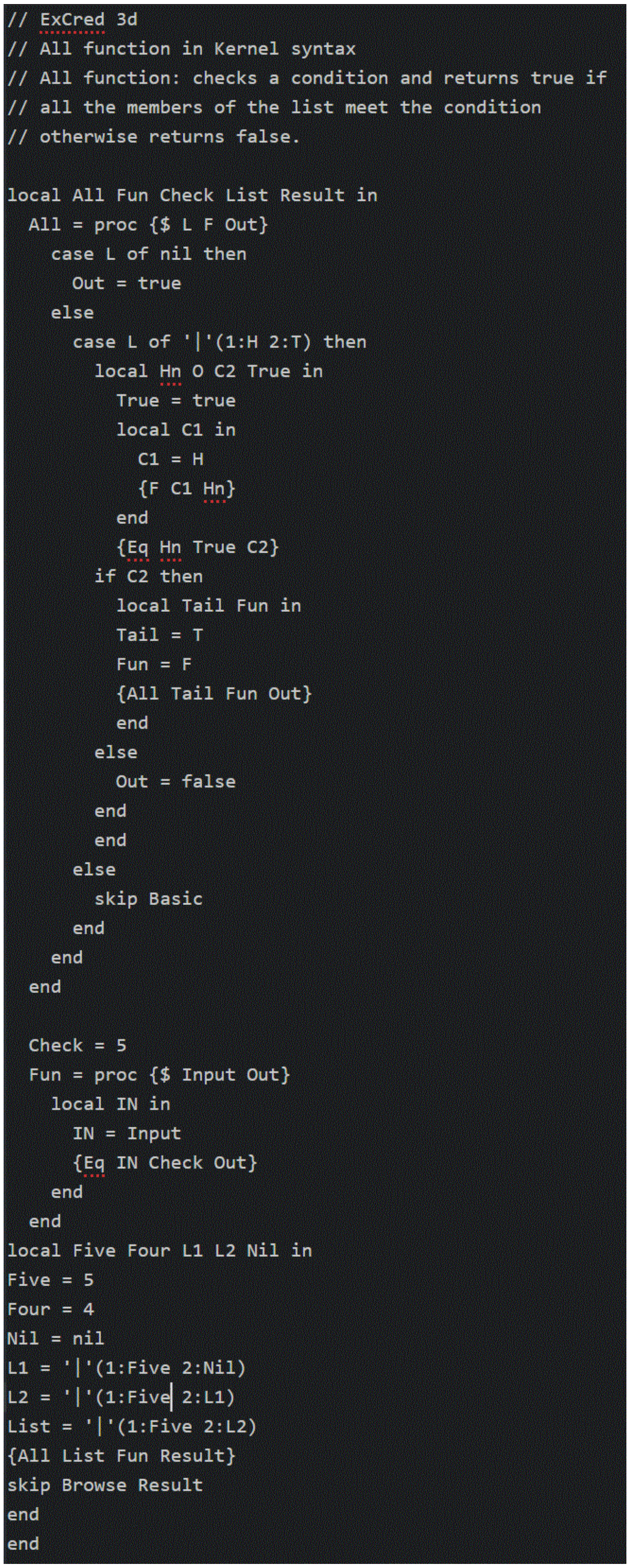
Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
//ExCred 3c
//num_nodes function for tree1
// in kernel syntax
// tree represented as record
//   empty
//   node(1:Element 2:LeftSubTree 3:RightSubTree)

local NumNds Five Six Mt Tree T1 Result in
  NumNds = proc {$ In Out}
    case In of empty then
      Out = 0
    else
      case In of node(1:N 2:L 3:R) then
        local C1 C2 Sum One in
          One = 1
          {NumNds L C1}
          {NumNds R C2}
          {IntPlus C1 C2 Sum}
          {IntPlus One Sum Out}
        end
      else
        skip Basic
      end
    end
  end
Five = 5
Six = 6
Mt = empty
T1 = node(1:Six 2:Mt 3:Mt)
Tree = node(1:Five 2:T1 3:Mt)
{NumNds Tree Result}
skip Browse Result
end
```

Eric Smrkovsky
9/30/2019
Csci 117
Lab6



```
// ExCred 3d
// All function in Kernel syntax
// All function: checks a condition and returns true if
// all the members of the list meet the condition
// otherwise returns false.

local All Fun Check List Result in
  All = proc {$ L F Out}
    case L of nil then
      Out = true
    else
      case L of '|'(1:H 2:T) then
        local Hn O C2 True in
          True = true
          local C1 in
            C1 = H
            {F C1 Hn}
          end
          {Eq Hn True C2}
        if C2 then
          local Tail Fun in
          Tail = T
          Fun = F
          {All Tail Fun Out}
          end
        else
          Out = false
        end
        end
      else
        skip Basic
      end
    end
  end

  Check = 5
  Fun = proc {$ Input Out}
    local IN in
      IN = Input
      {Eq IN Check Out}
    end
  end
local Five Four L1 L2 Nil in
Five = 5
Four = 4
Nil = nil
L1 = '|'(1:Five 2:Nil)
L2 = '|'(1:Five 2:L1)
List = '|'(1:Five 2:L2)
{All List Fun Result}
skip Browse Result
end
end
```