# PART1:

## // 1) nested if, nested case my kernel conversion

```
[
  local ["A","B"]
    [
      [local ["EXU1"] [EXU1=A,  = false()],
      local ["EXU2"] [EXU2 = true(),if EXU2 then [skip] elseif [local ["EXU3"] [EXU3 = B,if EXU3 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]]]
    ]
]
```

**Difference from actual kernel file:**

I created a local for A=false() while performing kernel conversion, i.e. [local ["EXU1"] [EXU1=A,  = false()] while this is not done in actual kernel conversion by compiler

Also, for skip Basic, actual compiler has done skip/BA  while in my conversion I wrote simple skip

also, for nested if-else ladder, I have used elseif as it is whereas in compiler conversion elseif is converted to else only.

## // 2) more expressions; note that applications of primitive binary operators

// ==, <, >, +, -, *, mod must be enclosed in parentheses (for hoz)

```
[
  local ["A"]
    [
     A = 2,local [local ["EXU2","EXU3"] [EXU2 = A,EXU3 = 1,"Eq" "EXU2" "EXU3" "EXU1"],if EXU1 then
[skip] else [skip]],
     local ["EXU1"]
      [
        local ["EXU2"] [EXU2 = A,local ["EXU5","EXU6"] [EXU5 = 3,EXU6 = 1,"IntMinus" "EXU5" "EXU6"
"EXU3"],"Eq" "EXU2" "EXU3" "EXU1"],
        if EXU1 then [skip]
      ]
    ]
]
```

**Difference from actual kernel file:**

First, I declared EXU1 directly in the block where EXU2, EXU3, Eq are written where as in actual compiler version its declared before local block of same. Similarly I did for EXU3 without declaring in local block of EXU2 I used it directly as a result holding variables

Second, skip basic is written as skip/BA in actual compiler conversion while I did skip only
Third, in the last if there is no else with if so I exclude the same but in actual compiler conversion else is also included by the compiler

## // 3) "in" declaration

```
[
 local ["X","Y"]
  [
     local ["EXU1","EXU2"] [EXU1 = 3,EXU2 = T,T = tree(1:EXU1 2:EXU2)],
        local ["A","B","PTU0"]
         [
           EXU4 = tree(1:A 2:B),EXU4 = 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]]]]
          ]

    ]
]
```

**Difference from actual kernel file:**

first, tree assignment is done wrong in my explanation
second, In my compiler explanation i forgot to create a local after local["X", "Y"] which is present in actual compiler conversion
third skip Browse is written as skip/BB in actual compiler conversion while I did skip only
fourth, after if block else was included in the actual compiler conversion but in my compiler conversion i didnt include the else part after if

## //4)expressions in place of statements

```
[
  local ["EXU1" ,"EXU2"][EXU1= Fun,EXU2=R]
  [
    Fun = proc {$ X EXU3} [EXU3 = X],
    local ["EXU3"] [EXU3 = 4,"Fun" "EXU3" "R"],
    skip
  ]
]
```

**Difference from actual kernel file:**

First, i declared EXU1 and EXU2 for "FUN" and "R" but in actual compiler conversion this was not done.
Second, for skip Browse I wrote skip only while in actual compiler convert skip Browse to skip/BR

## //5) Bind fun

```
[
 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 ["EXU6","EXU7"]
    [
      EXU6 = 5,local ["EXU8","EXU9"] [EXU8 = 3,EXU9 = 4,"IntMinus" "EXU8" "EXU9" "EXU7"],"IntPlus" "EXU6" "EXU7" "B"
    ],
    skip,
    skip,
    skip
  ]
]
```

**Difference from actual kernel file:**

for skip in the ending of kernel syntax I didn't mentioned like skip/BA, skip/s as it should be according to actual compiler conversion
for declaring EXU in compiler version, i declared EXU6, EXU7, EXU8, EXU9 where as in actual compiler conversion EXU2, EXU1, EXU3, EXU5, EXU5 have been re-used to save the extra memory

# PART2

## //A) 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
```

```
   L1 = (1|(2|(3|nil)))
   L2 = (4|(5|(6|nil)))

   Out = {Append L1 L2}
   skip Browse Out
   skip Full


   skip Basic
   skip Basic
   skip Basic
```

**// implementing reverse function**

```
   L3 = (6|(7|(8|(9|nil))))
   Reverse = fun{$ L}
             case L
               of nil then nil
               [] '|'(1:X 2:Xs) then {Append {Reverse Xs} [X]}
             end
          end
    Out1  = {Reverse L3}
    skip Browse Out1
    skip Full
end
```

/*

**output for only reverse function skip Browse**

Out1 : [ 9  8  7  6 ]

Store : ((106, 92), '|'(1:103 2:104)),
((105, 87, 74, 66, 65, 50), 9),
((104, 95), '|'(1:101 2:102)),
((103, 85, 69, 68, 48), 8),
((102, 98, 100, 97, 94, 54), '|'(1:90 2:91)),
((101, 77, 76, 46), 7),
((99, 78), nil()),
((96, 86, 82, 84, 81, 57), '|'(1:77 2:78)),
((93, 88, 79), '|'(1:85 2:86)),
((90, 89, 44), 6),
((91), nil()),
((83, 70), nil()),
((80, 75, 71, 73, 60), '|'(1:69 2:70)),
((72, 67), nil()),
((59, 63), '|'(1:66 2:67)),

((64, 51), nil()),
((62), nil()),
((61, 49), '|'(1:50 2:51)),
((58, 47), '|'(1:48 2:49)),
((56), '|'(1:74 2:75)),
((55, 45), '|'(1:46 2:47)),
((53), '|'(1:87 2:88)),
((52, 11), '|'(1:44 2:45)),
((43, 29), '|'(1:40 2:41)),
((42, 15), 1),
((41, 32), '|'(1:38 2:39)),
((40, 17), 2),
((39, 35, 37, 34, 31, 28, 10), '|'(1:21 2:22)),
((38, 19), 3),
((36, 20), nil()),
((33, 18), '|'(1:19 2:20)),
((30, 16), '|'(1:17 2:18)),
((27, 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","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)])),
((12), '|'(1:42 2:43)),
((13), proc(["L","EXU1"],[case L of nil() then [EXU1 = nil()] else [case L of '|'(1:X 2:Xs) then [local ["EXU2","EXU3"] [local ["EXU4"] [EXU4 = Xs,"Reverse" "EXU4" "EXU2"],local ["EXU4"] [EXU4 = X,local ["EXU5","EXU6"] [EXU5 = EXU4,EXU6 = nil(),EXU3 = '|'(1:EXU5 2:EXU6)]],"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 : ""

**Explanations of output:**

input L3 is [6,7,8,9]
output is printed by variable Out1

Store location of Out1 is 14, so binded value to store location 14 is ('|'(1:105 2:106))
store location 105 is having value '9' which is our first valye in reverse.
now store location 106 is binded to ('|'(1:103 2:104)), here store location 103 is binded to value '8' and this out second value in reversse
for store location 104 is binded to ('|'(1:101 2:102)) and store location 101 is having value '7' which is our third value in Reverse
for store location 102 is binded to ('|'(1:90 2:91)) and store location 90 is binded to value '6' which is 4th value in Reverse
and the store location 91 is binded to 'nil' and this marks the end of list and list is in reverse order now.
*/

## B) // Append with difference lists

```
local L1 End1 L2 End2 H1 T1 H2 T2 LNew Reverse Out L3 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
   skip Full
```

**//reverse diff function implementation as per professor suggestion during office hours**

```
Reverse = fun {$ Xs} Y1 ReverseD in
    proc {ReverseD Xs Y1 Y}
       case Xs
          of nil then Y1 = Y
          [] '|'(1:X 2:Xr) then {ReverseD Xr Y1 (X|Y)}
       end
    end
    {ReverseD Xs Y1 nil}
    Y1
 end
 L3 = (6|(7|(8|(9|nil))))


 Out = {Reverse L3}
 skip Browse Out
 skip Full

end
```

/*

**output**

```
*Hoz> runFull "declarative" "lab5-1/append_diff.txt" "lab5-1/append_diff.out"
Out : [ 9  8  7  6 ]

Store : ((18, 70, 65, 60, 55, 52, 49, 71), '|'(1:72 2:73)),
((73, 66), '|'(1:67 2:68)),
((72, 46), 9),
((69, 47), nil()),
((68, 61), '|'(1:62 2:63)),
((67, 44), 8),
((64, 45), '|'(1:46 2:47)),
((63, 56), '|'(1:57 2:58)),
((62, 42), 7),
((59, 43), '|'(1:44 2:45)),
((58, 53), nil()),
((57, 40), 6),
((54, 41), '|'(1:42 2:43)),
((51, 48, 19), '|'(1:40 2:41)),
((50), proc(["Xs","Y1","Y"],[case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],[("ReverseD",50)])),
((39, 27, 31, 11, 36, 15), Unbound),
((38, 8, 34), '#'(1:20 2:21)),
((21, 25, 9, 33, 13, 26, 35, 14), '|'(1:28 2:29)),
((10, 37), '#'(1:26 2:27)),
((20, 32, 12), '|'(1:22 2:23)),
((30), 4),
((28), 3),
((29), '|'(1:30 2:31)),
((24), 2),
((22), 1),
((23), '|'(1:24 2:25)),
((16), '#'(1:38 2:39)),
((17), proc(["Xs","EXU1"],[local ["Y1","ReverseD"] [ReverseD = proc {$ Xs Y1 Y} [case Xs of nil() then [Y1 = Y] else [case Xs of '|'(1:X 2:Xr) then [local ["EXU2","EXU3","EXU4"] [EXU2 = Xr,EXU3 = Y1,local ["EXU5","EXU6"] [EXU5 = X,EXU6 = Y,EXU4 = '|'(1:EXU5 2:EXU6)],"ReverseD" "EXU2" "EXU3" "EXU4"]] else [skip]]],local ["EXU2","EXU3","EXU4"] [EXU2 = Xs,EXU3 = Y1,EXU4 = nil(),"ReverseD" "EXU2" "EXU3" "EXU4"],EXU1 = Y1]],[])),
((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, "Out" -> 18, "L3" -> 19, "IntPlus" -> 1, "IntMinus" -> 2, "Eq" -> 3, "GT" -> 4, "LT" -> 5, "Mod" -> 6, "IntMultiply" -> 7)
Stack : ""

**EXPLANATION:**

Input given : L3= (6|(7|(8|(9|nil))))
Output Out = [ 9 8 7 6 ]

now, store location of Out is 18 which is binded to ('|'(1:72 2:73)) and store location 72 holds the value "9" which is our first value in reverse lists
store location 73 is binded to ('|'(1:67 2:68)), here store location 67 is binded to value "8" which is our second value in reverse list
store location 68 is binded to ('|'(1:62 2:63)), here store location 62 is binded to value "7" which is our third value in reveresed list
and store location 63 is bineded to ('|'(1:57 2:58)), here 57 store location is binded to value "6" and finally store location 58 is binded to "nil()", this marks the end of reverse lists

*/

## C) Count the number of cons operations '|' used to construct the output lists of Reverse in (A) vs (B) for a list of size 6. Explain your answer.

/***Ans.**

for Reverse (B) there were 16 cons operations while in Reverse(A) we saw total of 31 cons operations
there is a gap due to fact that, in Reverse(B) we are using itrative approach where we are doing any operation after the recusrive call of Reverse function
where as in Reverse(A) we are perforing append operation after the recusrive call which is leading to more cons operations.
*/