Saishnu Ramesh Kumar (300758706)

CSCI 117 – Lab 7

**Part 1:**

// Generate example from class

// First function multiplies all elements of a list by constant

local X Y Generate Display DisplayHamming in

fun {Generate N}

fun {$} (N#{Generate (N+1)}) end

end

proc {Display X N}

fun{DisplayHamming Z Num}

if(Num == 0) then nil

else

(V#F) = {Z} in

(V|{DisplayHamming F (Num - 1)})

end

end

local L in

L = {DisplayHamming X N}

skip Browse L

end

end

//Times Generator

local X Y Generate Display DisplayHamming Times in

fun {Generate N}

fun {$} (N#{Generate (N+1)}) end

end

fun {Times X Y}

fun {$}

(V#F) = {X} in

((V\*Y) # {Times F Y})

end

end

proc {Display X N}

fun{DisplayHamming Z Num}

if(Num == 0) then nil

else

(V#F) = {Z} in

(V|{DisplayHamming F (Num - 1)})

end

end

local L in

L = {DisplayHamming X N}

skip Browse L

end

end

//Merge Generator

local X Y Generate Display DisplayHamming Merge Times H in

fun {Generate N}

fun{$}(N#{Generate(N+1)}) end

end

fun {Times X Y}

fun {$}

(V#F) = {X} in

((V\*Y)#{Times F Y})

end

end

Merge = fun {$ X Y}

fun {$}

(V#F) = {X}

(U#H) = {Y} in

if (V < U) then (V#{Merge F Y})

else

if(V > U) then (U#{Merge X H})

else (V#{Merge F H})

end

end

end

H = fun {$} (1# {Merge {Times H 2}{Merge {Times H 3} {Times H 5}}})

end

proc {Display X N}

fun {DisplayHamming Z Num}

if(Num == 0) then nil

else

(V#F) = {Z} in

(V|{DisplayHamming F (Num-1)})

end

end

local L in

L = {DisplayHamming X N}

skip Browse L

end

end

fun{Take N G}

if(N <= 0) then []

else

(M#H) = {G} in

M|{Take N-1 H}

end

end

// Interleave example from class

fun {Zip X Y}

fun {$}

(V#F) = {X} in

(V#{Zip Y F})

end

end

// Testing

X = {Generate 3} // 3, 4, 5, ...

Y = {Generate 5} // 5, 6, 7, ...

Z = {Zip X Y} // 3, 5, 4, 6, 5, 7, ...

local

(V1#F1) = {Z}

(V2#F2) = {F1}

(V3#F3) = {F2} in

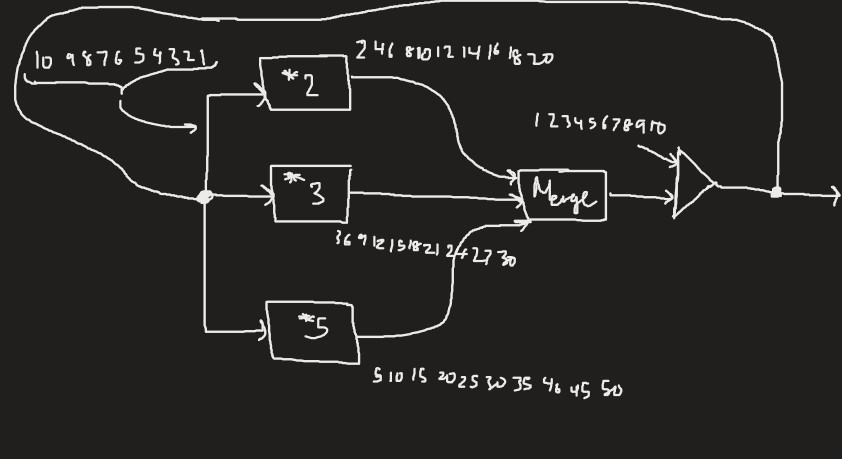
skip Browse V1 // 3 from X

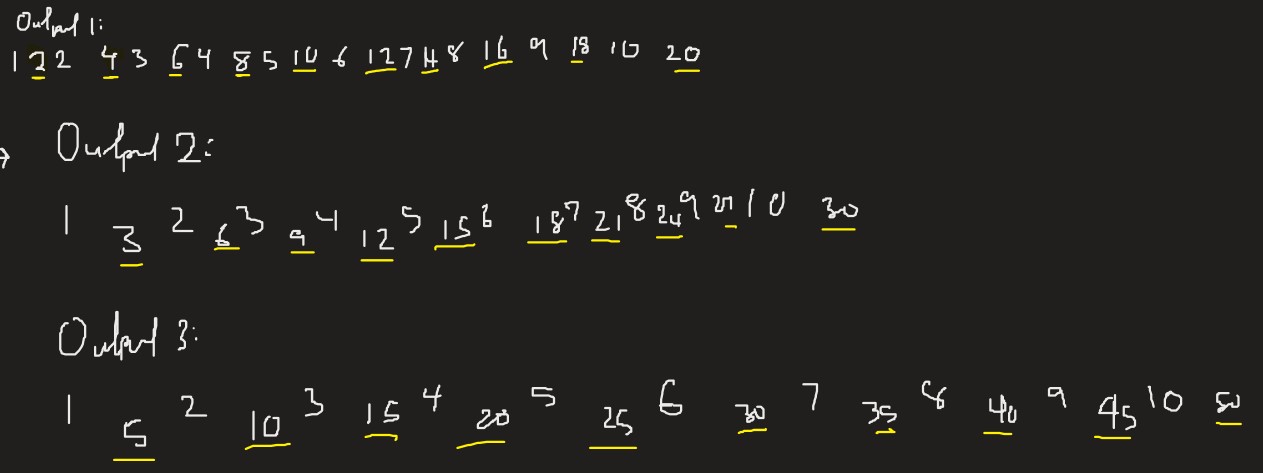
skip Browse V2 // 5 from Y

skip Browse V3 // 4 from X

end

**How these 10 values are produced:**





**Part 2:**

**Digital Logic:**

local GateMaker AndG OrG NotG A B S IntToNeed Out MulPlex in

fun {GateMaker F}

fun {$ Xs Ys} GateLoop T in

fun {GateLoop Xs Ys}

case Xs of nil then nil

[] '|'(1:X 2:Xr) then

case Ys of nil then nil

[] '|'(1:Y 2:Yr) then

({F X Y}|{GateLoop Xr Yr})

end

end

end

T = thread {GateLoop Xs Ys} end // thread isn't (yet) a returnable expression

T

end

end

fun {NotG Xs} NotLoop T in

fun {NotLoop Xs}

case Xs of nil then nil

[] '|'(1:X 2:Xr) then ((1-X)|{NotLoop Xr})

end

end

T = thread {NotLoop Xs} end // thread isn't (yet) a returnable expression

T

end

AndG = {GateMaker fun {$ X Y} if (X == 0) then 0 else (X\*Y) end end }

OrG = {GateMaker fun {$X Y} if (X == 1) then 1 else (X+Y) end end}

fun {IntToNeed L}

case L of nil then nil

[] '|' (1:X 2:Xr) then T W in

byNeed fun {$} X end W

T = {IntToNeed Xr}

(W|T)

end

end

fun {MulPlex A B S} R Z T W in

R = {NotG S}

Z = {AndG R A}

T = {AndG S B}

W = {OrG Z T}

W

end

A = {IntToNeed [0 1 1 0 0 1]}

B = {IntToNeed [1 1 1 0 1 0]}

S = [1 0 1 0 1 1]

Out = {MulPlex A B S}

// run a loop so the MulPlex threads can finish before displaying Out

local Loop in

proc {Loop X}

if (X == 0) then skip Basic

else {Loop (X-1)} end

end

{Loop 1000}

end

skip Browse Out

end

**Part 2a:**

fun {IntToNeed L}

case L of nil then nil

[]'|' (1:X 2:Xr) then T W in

byNeed fun {$} X end W

T = {IntToNeed Xr}

(W|T)

end

end

**Part 2b:**

AndG = {GateMaker fun {$ X Y} if (X == 0) then 0 else (X\*Y) end end }

OrG = {GateMaker fun {$X Y} if (X == 1) then 1 else (X+Y) end end}

**Part 2c:**

fun {MulPlex A B S} R Z T W in

R = {NotG S}

Z = {AndG R A}

T = {AndG S B}

W = {OrG Z T}

W

end

**Part 2d.1:**

The values for A and B are determined by the value S. If S is equal to zero, then it would not need both values A and B. If S is one, it would need the value of the other variables as well. For example, when A = 0, B = 1, and S = 1, then S will take the variables of A = 0, B = 1. If A, B, and S = 0, then S will automatically not take any values from the variables.

**Part 2d.2:**

Yes, they do match up with the results in Part2d.1.