import Data.Char

--questão 1

pertence n [] = False

pertence n (x:xs)= if n == x then True else pertence n xs

--questão 2

intersecao []\_ = []

intersecao \_[]= []

intersecao (x:xs) (y:ys) = if pertence x (y:ys) then x:intersecao xs (y:ys) else intersecao xs (y:ys)

--questão 3

inverso [] = []

inverso (x:xs)= inverso xs ++ [x]

--questão 4

nPrim 1 (x:xs) = [x]

nPrim n (x:xs) = x:nPrim (n-1) xs

nUltimos n xs = inverso (nPrim n(inverso xs))

--questão 5

soma2 [] \_ = []

soma2 \_[] = []

soma2 (x:xs) (y:ys)= x + y : soma2 xs ys

--questão 6

pot 0 = []

pot n = (2^n) : pot (n-1)

pot2 n = inverso (pot n)

--questão 7

intercalacao [] ys = ys

intercalacao xs [] = xs

intercalacao (x:xs) (y:ys) = if x> y then y: intercalacao (x:xs) (ys)

else if y>x then x:intercalacao (xs) (y:ys)

else x:y: intercalacao (xs) (ys)

--questão 8

menor [x] = x

menor (x:xs) = if x < menor (xs) then x else menor (xs)

--questão 9

removerElem n [] = []

removerElem n (x:xs) = if n == x then removerElem n xs else [x] ++ removerElem n xs

--questão 10

ordenar [] = []

ordenar (xs) = menor (xs) : ordenar( removerElem (menor xs) xs)

--questão 11

insereOrd n [] = [n]

insereOrd n (x:xs) = if n < x then n:(x:xs) else if n `elem` (x:xs) then (x:xs) else [x] ++ insereOrd n (xs)

--questão 12

enesimo \_ []= 0

enesimo 0\_ = 0

enesimo n (x:xs)= if n == 1 then x else enesimo (n-1) xs

--questão 13

repetir 0 \_ = []

repetir n e = e:(repetir (n-1) e)

--questão 14

removeTab [] = []

removeTab (x:xs) = if x == '\t' then ' ' : removeTab (xs) else [x]++ removeTab (xs)

--questão 15

minusculas [] =[]

minusculas (x:xs) = toLower x: minusculas xs

--questão 16

inversoDupla [] =[]

inversoDupla ((x,y):xys) =((y,x): inversoDupla xys)

--questão 17

simetrico [] = []

simetrico ((x,y):xys) = if x==y then (True : simetrico xys) else (False : simetrico xys)

--questão 18

menu [] =[]

menu (x:xs)|x==0 = '0':(menu xs)

|x==1 = '1':(menu xs)

|x==2 = '2':(menu xs)

|x==3 = '3':(menu xs)

|x==4 = '4':(menu xs)

|x==5 = '5':(menu xs)

|x==6 = '6':(menu xs)

|x==7 = '7':(menu xs)

|x==8 = '8':(menu xs)

|x==9 = '9':(menu xs)

numeros 0 = []

numeros n = (numeros (div (n - (rem n 10)) 10 )) ++ [rem n 10]

numString n = menu (numeros n)

--questão 19

num n = ord n +(-48)

string [] = 0

string (x:xs)= num x + 10\* string xs

stringNum xs = string(reverse xs)

--questão 20

decBin 0 = []

decBin n | rem n 2 == 0 = (decBin (div n 2))++['0']

| rem n 2 == 1 = (decBin (div n 2))++['1']

--questão 21

binDec "0" = 0

binDec "1" = 1

binDec x =

let

tamanho = length x

valorDigito ('0', \_) = 0

valorDigito ('1', exp) = 2^exp

in

sum [valorDigito(digito, tamanho - indice -1) | (indice, digito ) <- zip [0..] x]

--questão 22

troco x | div x 50 > 0 = (50,(div x 50)):(troco (x-(50\*(div x 50))))

| div x 20 > 0 = (20,(div x 20)):(troco (x-(20\*(div x 20))))

| div x 10 > 0 = (10,(div x 10)):(troco (x-(10\*(div x 10))))

| div x 5 > 0 = (5,(div x 5)):(troco (x-(5\*(div x 5))))

| otherwise = []

trocoCafe v d = troco (d-v)