

```

-----
--
-- (c) 2000 Alexander Vivian Hugh McPhail. All Rights Reserved
--
-----
module Marry (
  Candidate,
  Preferences,
  Marriage,
  --
  male, female,    -- Sex
  --
  makePreferences, -- [[Candidate]] -> [[Candidate]] -> Preferences
  findMatches     -- Preferences -> [Marriage]
) where
{-----}
import Set
import Array
import List
{-----}
type Sex = Int
type ID = Int

male, female :: Sex
male = 0
female = 1
{-----}
type Candidate = (Sex, ID)

type Marriage = (Candidate, Candidate)
{-----}
type Preferences = Array Candidate [Candidate]

type Married = Set Candidate
type Single = Set Candidate
{-----}
type Originator = Candidate
type Rank = Int
type PreviousBest = Rank

data RankedMarriage = RM Rank Originator Marriage
data MarriagePath = MP Originator Candidate [(PreviousBest, RankedMarriage)] Married Rank
{-----}
makePreferences :: [[Candidate]] -> [[Candidate]] -> Preferences
makePreferences mp fp
  | (length mp) == (length fp) = listArray ((male,1), (female, length mp)) (mp ++ fp)
  | otherwise                  = error "|men| != |women|"
{-----}

{-
pick a node
get preferred marriage
compute rank
update current best marriage
check for cycle
  get best marriage
  update sets
  split path
-}

{-----}
getSpouse :: Candidate -> Marriage
           -> Candidate
getSpouse c (h,w)
  | c == h    = w
  | otherwise = h
{-----}
getNextSpouse :: Candidate -> Preferences -> Married
              -> (Candidate, Preferences)
getNextSpouse c p m = check c p (p!c) m where
  check _ _ [] = error "getNextSpouse"
  check c p (x:xs) m
    | isElement x m = check c (p//[c,xs]) xs m -- destructive array assignment
    | otherwise     = (x,p)
{-----}
rankMarriage :: Candidate -> Candidate -> Preferences
              -> Rank
rankMarriage c1@(0,_) c2@(1,_) p = let rank c ps = strip (elemIndex c ps)
                                   strip (Just e) = e
                                   strip Nothing = error "rankMarriage:strip"

```

```

rankMarriage _ _ _ = error "rankMarriage"
{-----}
followChain :: Candidate -> Preferences -> Married ->
  (Candidate,RankedMarriage,Rank,Preferences)
followChain c@(s,i) p m = let (spouse,preferences) = getNextSpouse c p m
  (husband,wife) = if (s == 0)
    then (c,spouse)
    else (spouse,c)
  rank = rankMarriage husband wife p
  in (spouse,(RM rank c (husband,wife)),rank,preferences)
{-----}
extendChain :: Candidate -> Preferences -> Married
  -> (MarriagePath,Preferences)
extendChain c p m = let (spouse,marriage,rank,preferences) = followChain c p m
  new_married = (insertSet spouse (singletonSet c))
  in extendChain' (MP c spouse [(-1,marriage)] new_married rank)
  preferences m new_married

extendChain' :: MarriagePath -> Preferences -> Married -> Married
  -> (MarriagePath,Preferences)
extendChain' mp p m nm = fst (extendChain'' mp p m nm)

extendChain'' :: MarriagePath -> Preferences -> Married -> Married
  -> ((MarriagePath,Preferences),Married)
extendChain'' (MP fc lc rm _ r) p m nm = let (spouse,marriage,rank,preferences) = followChain lc p m
  new_rank = if (r > rank) then rank else r
  new_rm = (r,marriage):rm
  new_nm = insertSet spouse nm
  in if (isElement spouse nm)
    then (((MP fc spouse new_rm new_nm
  new_rank),preferences),nm)
    else extendChain'' (MP fc spouse new_rm new_nm new_rank)
  preferences m new_nm
{-----}
splitChain :: MarriagePath
  -> Marriage
splitChain (MP _ _ rm _ ra) = check rm ra where
  check [] ra = error "splitChain"
  check ((pb,RM r _ m):rms) ra
    | r == ra = m
    | otherwise = check rms ra
{-----}
findMatch :: Preferences -> Single -> Married
  -> (Marriage,Preferences)
findMatch p s m = let (path,preferences) = extendChain (first s) p m
  in (splitChain path,preferences)

findMatches :: Preferences -> [Marriage]
findMatches p = doFind p (listToSet (indices p)) emptySet where
  doFind p s m = let ((husband,wife),preferences) = findMatch p s m
  new_married = (insertSet husband (insertSet wife m))
  new_single = (removeSet husband (removeSet wife s))
  in if (isEmptySet new_single) then [(husband,wife)]
  else [(husband,wife)] ++ (doFind preferences new_single
new_married)
{-----}

```