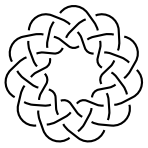
```
{-# OPTIONS -W #-}
module Main where
import Crossing
import Control.Monad          (replicateM_, forM_)
import System.Environment     (getArgs)
import System.FilePath        (splitExtension)
```

1. Examples

Here is a simple collar:

```
diagonals = ⟨√2, √2⟩
            ⟨√2, -√2⟩
collarUnit      :: M_ t ()
collarUnit      = replicateM_ 11
                  $ do crossing 2 ↘ ⟨-5, -5⟩ diagonals
                       crossing   1 ↗ ⟨0, 0⟩ diagonals
                       crossing     0 ↘ ⟨5, 5⟩ diagonals
                       advance ⟨10, 0⟩
collarControls :: M_ Paint ()
collarControls = begin 0 (red Open) ≫ collarUnit ≫
                   replicateM_ 3 (begin 0 (paint Open) ≫ collarUnit)
collar :: OC → [Maybe (Thread_ Paint)]
collar Open   = weave collarControls 4
collar Closed = weave (begin 0 (paint Closed) ≫ circular collarUnit) 4
```



More generally, here is a Turk's Head. The two paramaters are k, the number of strands, and n, the number of crowns in a loop around the collar. If k and n are not relatively prime, this code doesn't currently work.

```
turksHead :: Int → Int → OC → [Maybe (Thread_ Paint)]
turksHead k n oc =
  let spacing  = 10
      stepping = 5
      unit     :: M_ t ()
      unit     = replicateM_ n
                 $ forM_ [k - 2, k - 3 .. 0] (λi.
                     let x = spacing/2 - stepping × fromIntegral i
                         y = stepping × (fromIntegral (k - 2)/2 - fromIntegral i)
                     in crossing i (if even i then ↘ else ↗) ⟨x, y⟩ diagonals)
                 ≫ advance ⟨spacing, 0⟩
```

```
    controls :: M_ Paint ()
    controls = begin 0 (red Open) ≫ unit ≫
               replicateM_ (k − 1) (begin 0 (paint Open) ≫ unit)
  in case oc of Open   → weave controls k
                Closed → weave (begin 0 (paint Closed) ≫ circular unit) k
```

The fishbone braid. Quite similar, except for the signs of the crossings. This braid has $2k + 1$ total strands.

```
fishbone :: Int → Int → OC → [Maybe (Thread_ Paint)]
fishbone kk1 n oc | kk1 > 2 ∧ odd kk1 =
  let k        = kk1 `div` 2
      spacing  = 10
      stepping = 5
      unit     :: M_ t ()
      unit     = replicateM_ n
               $ forM_ [2 × k − 1, 2 × k − 2 .. 0] (λi.
                   let x = spacing/2 − stepping × fromIntegral i
                       y = stepping × (fromIntegral k − 1/2 − fromIntegral i)
                   in crossing i (if i < k then ⤡ else ⤢) ⟨x, y⟩ diagonals)
               ≫ advance ⟨spacing, 0⟩
      controls :: M_ Paint ()
      controls = begin 0 (red Open) ≫ unit ≫
                 replicateM_ (2 × k) (begin 0 (paint Open) ≫ unit)
  in case oc of Open   → weave controls kk1
                Closed → weave (begin 0 (paint Closed) ≫ circular unit) kk1
fishbone kk1 _ _ = error (show kk1 ++ " is not an odd integer > 2")
```

Celtic knot 1, another braid, with 4 strands. This one uses two pieces of rope.

```
celtic1 :: Int → OC → [Maybe (Thread_ Paint)]
celtic1 n oc =
  let unit     :: M_ t ()
      unit     = replicateM_ n
               $ do crossing 2 ⤢ ⟨0, −5⟩ diagonals
                    crossing   1 ⤡ ⟨5, 0⟩ diagonals
                    crossing     0 ⤢ ⟨10, 5⟩ diagonals
                    crossing   1 ⤡ ⟨15, 0⟩ diagonals
                    advance ⟨20, 0⟩
      controls :: M_ Paint ()
      controls = begin 0 (orange Open) ≫ unit ≫
                 replicateM_ 2 (begin 0 (paint Open) ≫ unit) ≫
                 end 0 (orange Open) ≫
                 begin 0 (paint Open) ≫ begin 1 (red Open) ≫ unit
```

```
  in case oc of Open   → weave controls 4
                Closed → weave (begin 0 (paint Closed) ≫
                                begin 1 (red Closed) ≫ circular unit) 4
```

A wider version, with 6 strands.

```
celtic2 :: Int → OC → [Maybe (Thread_ Paint)]
celtic2 n oc =
  let unit     :: M_ t ()
      unit     = replicateM_ n
               $ do crossing 4 ⤮ ⟨−10, −10⟩ diagonals
                    crossing   3 ⤮ ⟨−5, −5⟩ diagonals
                    crossing     2 ⤮ ⟨0, 0⟩ diagonals
                    crossing       1 ⤮ ⟨5, 5⟩ diagonals
                    crossing         0 ⤮ ⟨10, 10⟩ diagonals
                    crossing   3 ⤮ ⟨5, −5⟩ diagonals
                    crossing       1 ⤮ ⟨15, 5⟩ diagonals
                    advance ⟨20, 0⟩
      controls :: M_ Paint ()
      controls = begin 0 (orange Open) ≫ unit ≫
                 replicateM_ 3 (begin 0 (paint Open) ≫ unit) ≫
                 end 0 (orange Open) ≫
                 begin 0 (paint Open) ≫ begin 1 (red Open) ≫ unit ≫
                 end 1 (red Open) ≫
                 begin 1 (paint Open) ≫ begin 3 (red Open) ≫ unit
  in case oc of Open   → weave controls 6
                Closed → weave (begin 0 (paint Closed) ≫
                                begin 1 (red Closed) ≫
                                begin 3 (red Closed) ≫ circular unit) 6
```

```
celtic3 :: Int → OC → [Maybe (Thread_ Paint)]
celtic3 n oc =
  let unit     :: M_ t ()
      unit     = replicateM_ n
               $ do crossing 4 ⤮ ⟨−10, −10⟩ diagonals
                    crossing   3 ⤮ ⟨−5, −5⟩ diagonals
                    crossing     2 ⤮ ⟨0, 0⟩ diagonals
                    crossing       1 ⤮ ⟨5, 5⟩ diagonals
                    crossing         0 ⤮ ⟨10, 10⟩ diagonals
                    crossing   3 ⤮ ⟨5, −5⟩ diagonals
```

```
                crossing      2 ↘ ⟨10, 0⟩ diagonals
                crossing        1 ↘ ⟨15, 5⟩ diagonals
                advance ⟨20, 0⟩
    controls :: M_ Paint ()
    controls = begin 0 (orange Open) ≫ unit ≫
               replicateM_ 3 (begin 0 (paint Open) ≫ unit) ≫
               end 0 (orange Open) ≫
               begin 0 (paint Open) ≫ begin 1 (red Open) ≫ unit ≫
               begin 1 (paint Open) ≫ unit
  in case oc of Open   → weave controls 6
                Closed → weave (begin 0 (paint Closed) ≫
                                begin 1 (red Closed) ≫ circular unit) 6
```

```
dragonTamer :: Int → OC → [Maybe (Thread_ Paint)]
dragonTamer n oc =
  let unit     :: M_ t ()
      unit     = replicateM_ n
               $ do crossing 5 ↘ ⟨−10, −10⟩ diagonals
                    crossing   4 ↘ ⟨−5, −5⟩ diagonals
                    crossing     3 ↘ ⟨0, 0⟩ diagonals
                    crossing       2 ↘ ⟨5, 5⟩ diagonals
                    crossing         1 ↘ ⟨10, 10⟩ diagonals
                    crossing           0 ↘ ⟨15, 15⟩ diagonals
                    advance ⟨10, 0⟩
      controls :: M_ Paint ()
      controls = begin 0 (red Open) ≫ unit ≫
                 replicateM_ 6 (begin 0 (paint Open) ≫ unit)
  in case oc of Open   → weave controls 7
                Closed → weave (begin 0 (paint Closed) ≫ circular unit) 7
```

2. Main program

This program takes any number of file names on the command line, each ending in `.svg` or `.tex`. It figures out which SVG or TikZ files to produce by parsing the file name.

```
main :: IO ()
main = getArgs ≫= mapM_ (λfp. writeFile fp $! contents fp)

contents :: FilePath → String
contents fp = case splitExtension fp of
                (fp, ".svg") → svg  (contentThreads fp)
                (fp, ".tex") → tikz (contentThreads fp)
                _            → error ("Don't know how to make " ++ fp)
```

```
data Contents = Collar                 OC
              | Turks          Int Int OC
              | Fish           Int Int OC
              | Celtic1        Int     OC
              | Celtic2        Int     OC
              | Celtic3        Int     OC
              | DragonTamer    Int     OC deriving Read
contentThreads :: String -> [Maybe (Thread_ Paint)]
contentThreads s = case read (map sp s) of
                     Collar              oc -> collar              oc
                     Turks          k n  oc -> turksHead      k n  oc
                     Fish           kk1 n oc -> fishbone      kk1 n oc
                     Celtic1        n    oc -> celtic1        n    oc
                     Celtic2        n    oc -> celtic2        n    oc
                     Celtic3        n    oc -> celtic3        n    oc
                     DragonTamer    n    oc -> dragonTamer    n    oc
  where sp '_' = ' '
        sp c   = c
```