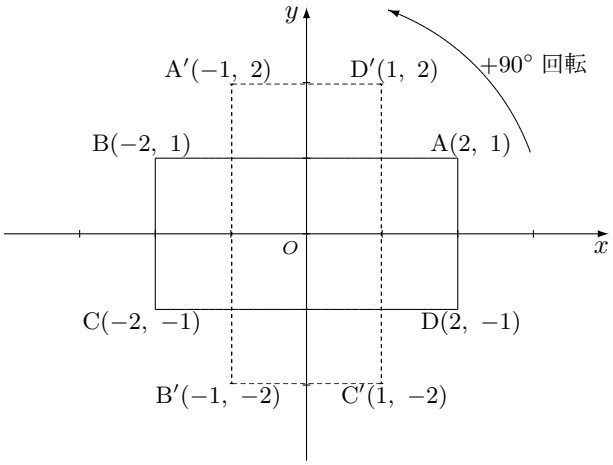
ドラゴンカーブ

中 2–中 3 にかけて、回転や拡大・縮小について学んだことを思い出してほしい。それらを Swift で実現するには三角関数などの知識が用いられ、実際にプログラムを書くハードルは結構高かったのではないだろうか。たしかに一般論を述べれば高校数学へ立ち入らざるを得ない。しかし用途を絞ればその限りではない。



図は、長方形 ABCD が原点を中心に、+90° 回転して長方形 A′B′C′D′ へ移ったところである[1]。これにより各頂点は

$$\mathrm{A}(2,\ 1) \to (-1,\ 2),\quad \mathrm{B}(-2,\ 1) \to (-1,\ -2),\quad \mathrm{C}(-2,\ -1) \to (1,\ -2),\quad \mathrm{D}(2,\ -1) \to (1,\ 2)$$

と移動している。これだけを見て一般の規則を述べるのは少し乱暴かもしれないが

$$\mathrm{P}(a,\ b) \quad \to \quad \mathrm{P}'(-b,\ a)$$

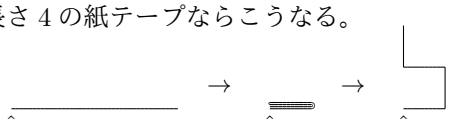
となっていることがわかるだろう。しかし、このことは実際正しい。理由は、90° の回転とは『座標軸ごと 90° 回転した図を、新たな $\dot{x}$-$\dot{y}$ 座標として見る』ことだからだ。

[1] 数学では、反時計回りが正の回転、時計回りが負の回転としている。

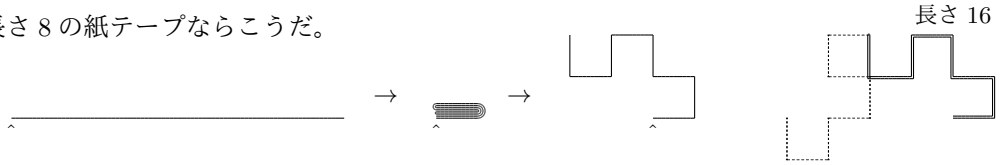
図を見てわかるように、$\dot{y}$ 座標の目盛は x 座標の目盛と一致し、$\dot{x}$ 座標の目盛は y 座標の目盛と逆向きである。これは $(\dot{x},\ \dot{y}) = (-y,\ x)$ を意味する。

さて、この回転の様子が何になるかというと、たとえば細長い紙テープの折りたたみに関してである。具体的に長さ 2 の紙テープを半分に折りたたんだあと、折り目が 90° になるように広げることを考えよう。長さ 2 の紙テープなら、こういうことである（^ は左端の目印）。

長さ 4 の紙テープならこうなる。



長さ 8 の紙テープならこうだ。



そして長さ 16 の紙テープなら、長さ 8 の紙テープからもう一度広げることができるように折りたたまれているだろう。それを広げれば、“長さ 8 を広げた形” が 90° 曲げて複製されるはずだ。

この作業は、長さ 32, 64, 128, ... と延々続けることができる。このとき見られる形が**ドラゴンカーブ**と呼ばれるものだ。名前の由来がはっきりするのは、もっと先まで続けたときである。見たくなってきたかな？

だったらプログラムを書くしかないね。その際ポイントは『それを広げれば、... 複製されるはずだ。』と書いたところだ。これは、一つ前の状態を利用できることを意味する。つまり、再帰によるプログラムが書けるということである。そして 90° の回転は、冒頭に述べた仕組みで記述できるじゃないか。

[dragoncurve.playground]

```
 1 import SwiftUI
 2
 3 struct ContentView: View {
 4     var body: some View {
 5         Path { path in
 6             @State var FOLDNUM = 10
 7             @State var UNITLEN = 50 / FOLDNUM
 8             @State var arrnode = locations(v: vectors(n: FOLDNUM))
 9
10             for i in 0 ..< (arrnode.count-1) {
11                 path.move(   to: CGPoint(x: UNITLEN * arrnode[i  ].0,
                                            y: UNITLEN * arrnode[i  ].1))
```

```
12                 path.addLine(to: CGPoint(x: UNITLEN * arrnode[i+1].0,
                                            y: UNITLEN * arrnode[i+1].1))
13             }
14         }
15         .stroke(lineWidth: 1)
16         .fill(Color.red)
17         .frame(width: 1, height: 1)
18     }
19
20     func vectors(n: Int) -> [(Int, Int)] {
21         if n > 0 {
22             var v = vectors(n: n-1)
23             for i in (0 ..< v.count).reversed() {
24                 v.append((-v[i].1, v[i].0))
25             }
26             return v
27         } else {
28             return [(1, 0)]
29         }
30     }
31
32     func locations(v: [(Int, Int)]) -> [(Int, Int)] {
33         var l = [(0, 0)]
34         for i in 0 ..< v.count {
35             l.append((l[i].0 + v[i].0, l[i].1 + v[i].1))
36         }
37         return l
38     }
39 }
```

紙テープを折って広げるシミュレーションをする割には、短いプログラムだと思うだろう。再帰関数を利用した恩恵である。ただ、ちょっと変なこともしている。

変なことの理由は、先の紙テープの “折り広げ図” を見ればわかる。図に示した折り方は、左端を固定して右側から左へ左へ...と半分に折りたたんでいる。そして長さ 1 になったら、今度は右側へ向けて広げるのだが、広げるときの “支点” は右側—直前の折り目—であり、その位置は広げるたびに上下左右へずれていく。これは当然の作業であるが、先に述べた回転の仕組みは原点を支点に回転させるのであった。すると、この状態は広げるたびに原点があちらこちらへずれて具合が悪い。なぜなら、回転する支点の座標がずれるからだ。

そこで、回転に関する再帰関数を座標ではなく**ベクトル**で、つまり『どの方向に延びるか』を求めるだけにした。それが `func vectors(n: Int) -> [(Int, Int)]` だ。この関数は整数値 `n`（紙テープを折る回数）を受け取って、紙テープの折り目から先のベクトルの方向 `v` を返す関数である。したがって初期値 `[(1, 0)]` は、長さ 1 の線分が座標 $(0,\ 0)$ から $(1,\ 0)$ に向かって置かれて

いると想定している。単に (1, 0) を返すのではなく [(1, 0)] であるのは、ベクトルの配列としてデータを保持したいためである。

v = vectors(n: n-1) が再帰を用いる前の準備といったところで、一つ前の状態が変数 v に与えられる。ここに次の状態を追加すれば、一つ前の状態から次の状態を求めたことになる。それが直後の for 文で、v.append((-v[i].1, v[i].0)) により順に $(-y,\ x)$ のベクトルが追加されることがわかるだろう。このとき、格納されるベクトルの値は (±1, 0) か (0, ±1) のいずれかとなっている。

ここで.reversed() メソッドを用いて、vectors の最後の要素から逆向きに追加していることに注意しよう。理由は、先の “長さ 16” の図を見てわかるように、複製されてつながる線は折り返しから逆向きに延びているからだ。

以上で、配列 v には折り目から延びるベクトルが原点に近いものから順に格納された。それらは、正しい座標に変換しないと図が描けない。そこで、座標を正す関数 locations が必要になった。ここを通して arrnode に格納された座標が、実際の x-y 座標の位置である。それは、直前の座標にベクトルを直（じか）に加えることで、次の折り目の位置を求めている。初期値は原点である (0, 0) だ。

また、描画に際して単位長が 1 だと困るので、相応の倍率 UNITLEN を指定してある。折りたたみ回数が少ないときは単位長は長くてもよいけれど、折りたたみ回数が増えれば図が画面からはみ出る可能性がある。そのため大雑把ではあるが、FOLDNUM の大きさで UNITLEN を調整した。あとは path をつなげればよい。それはテキストの表示ではないので、VStack { は Path { に変わっていることに注意されたい。

SwiftUI では、.stroke()、.fill()、.frame() などのメソッドで線の幅等を決めるのだが、.frame の幅と高さを 1 にしているのは違和感があるはずだ。本来なら、たとえば 200×400 などとして描画範囲を指定するものである。すると SwiftUI は、自動的に描画範囲を機器の画面中央に配置してくれる。しかし、そうすると 200×400 サイズの左上が座標 (0, 0) となるので、(100, 200) の位置を原点に指定し直す必要が生じる。そこで、SwiftUI が描画を機器の画面中央に表示することを逆手に、描画サイズを 1×1 にすれば、原点がほぼ画面中央になるということである。なるべく余分なコードを記述しないように考えたことだが、明らかに不当な処置であることは否（いな）めない。ここだけの話にしてもらえば幸いである。

一番自然な方法は、.frame のサイズに機器の画面サイズ (WIDTH, HEIGHT) を与えて、原点を (WIDTH/2, HEIGHT/2) に変換することだろう。また、コードは折り広げ図をもとに書いているが、

この通り実行すると折り広げ図とは上下が逆に表示されるだろう。それは、数学の y 座標は『上方向が正、下方向が負』であるが、SwiftUI の y 座標は『左上が $(0,\ 0)$、下方向が正』となっているからである。そのため、折り広げ図のように表示させるには、y 方向の正負を逆にする必要もある。そのための関数は各自で用意してもらおう。