# 課題 1

カタラン数を計算する関数 `catalan n` を定義しなさい。

## 解答及び解説

問題文にある、次の漸化式を計算すると良い。

$$c_n = c_0c_{n-1} + c_1c_{n-2} + \cdots + c_{n-1}c_0$$

一般的な繰り返しを再帰関数で表現するとき、例えば次のようにする。

$$loop\ i\ n\ =\ \texttt{if}\ i == n\ \texttt{then}\ END\ \texttt{else}\ EACH(i)\ \star\ (loop\ (i+1)\ n)$$

今の処理が何度目 $(i)$ であるかと、全部で何回 $(n)$ 繰り返すかの二つの値を引き数とする。(各回での値 $(EACH(i))$ と残り $(loop\ (i+1)\ n)$ の間で、何らかの演算 $(\star)$ を行う。)

今回の場合、漸化式の各項は、二つのカタラン数の掛け算になっており、一方の添え字が 0 から $(n-1)$ まで増加し、もう一方が逆に $(n-1)$ から 0 まで減少している。このことをうまく利用すると、次のようなプログラムが考えられる。

```
catalan :: Integer -> Integer
catalan 0 = 1
catalan n = catalan' 0 (n-1)

catalan' :: Integer -> Integer -> Integer
catalan' i 0 = (catalan i * catalan 0)
catalan' i j = (catalan i * catalan j) + catalan' (i+1) (j-1)
```

この場合、終了条件が「引き数が 0 になる」ときなので、パターンを使って書くことが出来て、とても分かりやすい。(出題の時期を考えると、パターンを使った書き方ではなく、`if-then-else` を使った書き方にすべきですが、実際にやってみると分かる通り複雑でわかりにくくなってしまう。) ちなみに繰り返しのインデックスが減少するのでも良い場合は、以下のようなパターンを使った書き方の方がわかりやすくて良い。

$$\begin{aligned} loop\ 0 &= END \\ loop\ i &= EACH(i)\ \star\ loop\ (i-1) \end{aligned}$$

しかしながら、この方法でカタラン数を求めるには、再帰的な計算が二重、三重に繰り返されるため、あまりに効率が悪い。少しでも効率を良くするために、漸化式の最初の項と最後の項が同じであることを利用して、次のようにすることも考えられる。

```
catalan' :: Integer -> Integer -> Integer
catalan' i j | i == j    = (catalan i)^2
             | i > j     = 0
             | otherwise = 2*(catalan i * catalan j) + catalan' (i+1) (j-1)
```

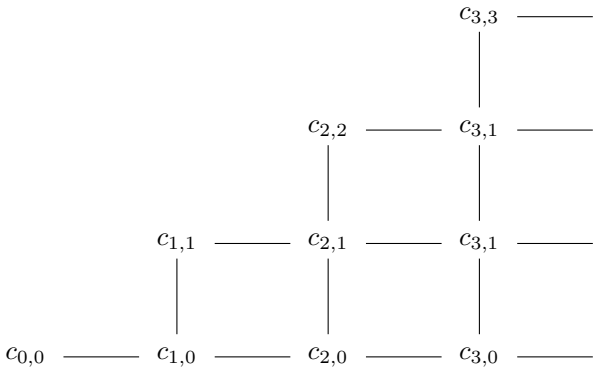
この方法では、同じ値の項が二つずつあるので、計算する項をおよそ半分に減らせる。但し、項数が奇数の場合、真ん中の項は一つしかないことに注意が必要である。(上記のプログラムでは、項数が偶数ならiとjはすれ違い、奇数ならiとjが同じになることを利用している。)

さて、課題の意図に沿って、

「再帰関数を使って、漸化式の計算や繰り返しの計算をする方法を考える。」

という方針で解答すると、以上のようになる。

しかし他にも、カタラン数を求めるには、様々な方法が考えられる。次のプログラムでは、問題文中の図の対角線上だけでなく、各格子点について、そこへ到達する道順が何通りあるかを順次計算する方法をとっている。




図中の各格子点に、座標を割り当てる。左下が (0,0) で、その右隣を (1,0)、その上が (1,1) という具合に。すると格子点は $(n, m)$ (但し $n \geqq m$) となる。各格子点に到達する道は左隣から来るものと、下から来るものの二通りがあるので、その 2 点までの道筋の数を足し合わせればその格子点への道筋の数になる。最下段 (n, 0) には、左隣から来るのみで、ずっとその道を辿って来るため一通りしかない。

```
cat n 0 = 1
```

対角線上の点 (n, n) は、下からしか来ない。

```
cat n m = if n == m then cat n (m-1)
```

他は、両方から来るので、

```
        else cat (n-1) m + cat n (m-1)
```

となる。

```
catalan :: Integer -> Integer
catalan n = cat n n

cat :: Integer -> Integer -> Integer
cat n 0 = 1
cat n m = if n == m
    then cat n (m-1)
    else cat (n-1) m + cat n (m-1)
```

さて、もっと効率を重視した方法もある。まずは、リストを使った次のような解答。

```
catalan :: Integer -> Integer
catalan n = x
    where x:xs = catalan' n

catalan' :: Integer -> [Integer]
catalan' 0 = [1]
catalan' n = x:xs
    where xs = catalan' (n-1)
          x = catalan'' (reverse xs) xs

catalan'' :: [Integer] -> [Integer] -> Integer
catalan'' [] [] = 0
catalan'' (x:xs) (y:ys) = (x*y) + (catalan'' xs ys)
```

関数 catalan' は、n+1 個のカタラン数のリストを作る。そして、関数 catalan'' は、$c_n$ を計算するのに、catalan' (n-1) を計算することにより得られたリスト

$$[c_{n-1}, c_{n-2}, \ldots, c_1, c_0]$$

と、それを逆順にしたものとで、各要素を掛け合わせて、それらの合計を求める。

実際に、リストに対するこのような操作は、次のようにする方が、Haskell らしいプログラムとなる。

```
catalan'' xs ys = sum $ map mul $ zip xs ys
    where mul (s, t) = s * t
```

最後に、公式 (?) を使った次のような計算方法もある。

```
catalan :: Integer -> Integer
catalan n = div (fact (2*n)) ((fact (n+1))*(fact n))
```

この方法については、出題時には予期しておらず、私の出題ミスです。出題意図とはずれた解答ですが、この解答も、もちろん正解としました。

今後は、カタラン数はやめて、再帰的な関数で求められるもう少し別の数列を探すことにします。