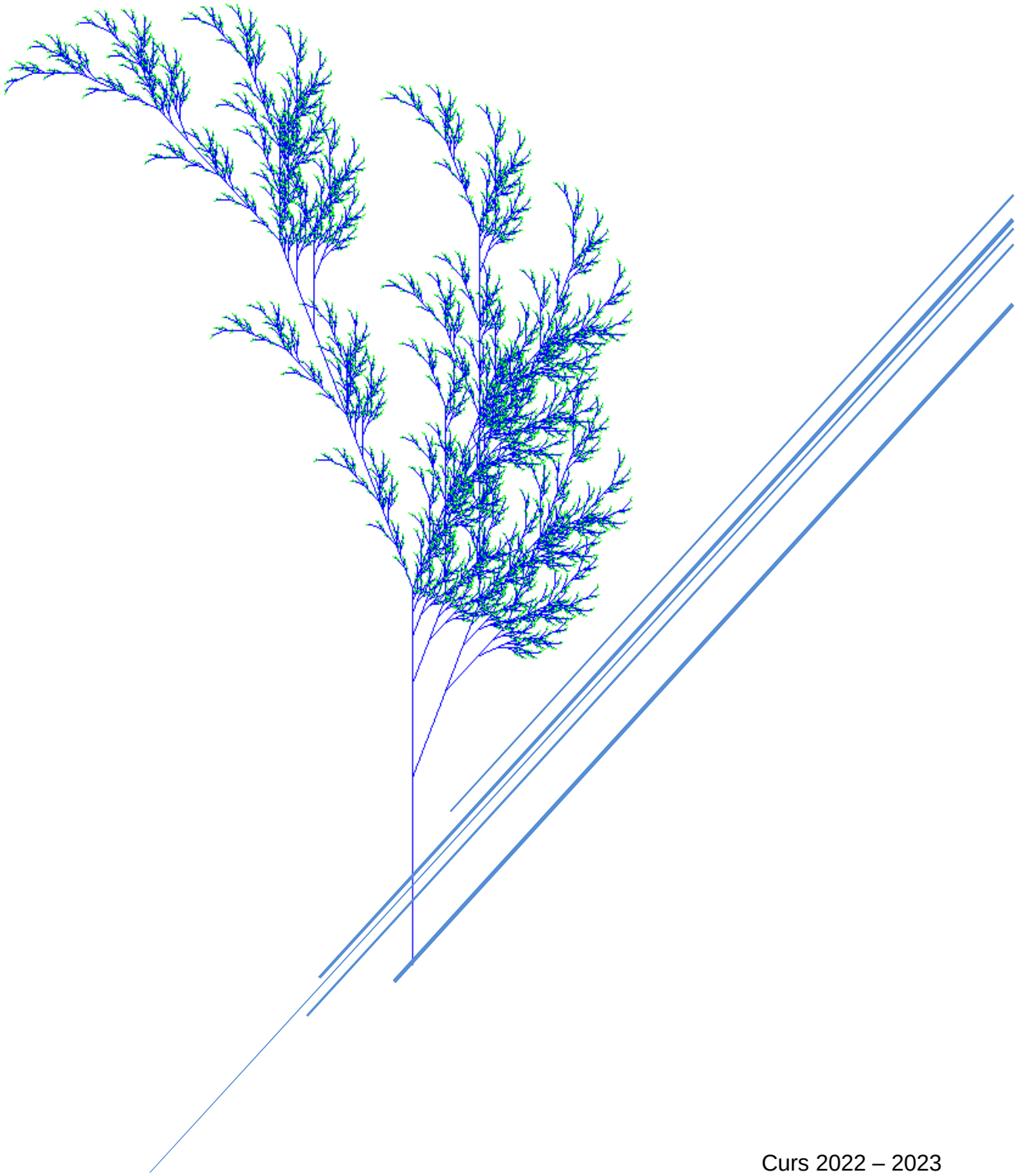
John, l'Artista

Paradigmes i Llenguatges de Programació



Curs 2022 – 2023

Raúl Lupiáñez Miranda (u1973789)
Guillem Trabal Zafra (u1960297)

Index

Descripció problema i solució

Per aquesta pràctica, se'ns ha plantejat la possibilitat de poder dibuixar figures amb Haskell. Si bé és cert que bona part del codi ja se'ns era proporcionada, hi havia parts que faltaven, o simplement fèiem diferents activitats per anar familiaritzant-nos a fer servir Haskell, per després fer dibuixos amb el programa. Finalment, arribarem a dibuixar fins i tot fractals.

El nostre codi permet realitzar totes les tasques demanades, a més a més de poder dibuixar fractals específics (mentre es tingui la gramàtica sempre es poden dibuixar més fractals, ja que és fàcil de programar, de fet, s'han programat dos fractals més amb les seves pròpies gramàtiques, es podran veure al fitxer Main.hs).

Particularitats del codi

En aquesta secció parlarem dels diferents problemes plantejats per la realització de la pràctica, així com mostrar el seu codi i resultats d'alguns dels jocs de proves.

Problema 1. separa

```
separa :: Comanda -> [Comanda]
```

Aquesta funció rep una Comanda com a entrada i retorna una llista de Comandes de comandes individuals (sense que cap Comanda tingui el valor **Comanda :#: Comanda**) sense cap comanda **Para**.

La funció separa té tres casos:

- El primer cas és quan la Comanda d'entrada és **Para**. En aquest cas, la funció retorna una llista buida ([]) per tal de no afegir-la a la llista resultant.
- El segon cas és quan la Comanda d'entrada és una composició de dues Comandes, indicada per l'operador **:#:**. En aquest cas, la funció crida recursivament separa en cadascuna de les dues Comandes i concatena les llistes resultants utilitzant l'operador **++**.
- El tercer cas és quan la Comanda d'entrada no és una composició de dues Comandes, és a dir, és una Comanda individual (per exemple **Avança d** on 'd' és la distància a avançar). En aquest cas, la funció retorna una llista que conté la Comanda d'entrada.

En general, la funció separa s'utilitza per separar una Comanda en les seves components individuals. Si la Comanda és una composició de dues Comandes, la funció separa recursivament cada component i retorna una llista de totes les Comandes individuals. Si la Comanda no és una composició de dues Comandes, la funció simplement retorna una llista que conté la Comanda d'entrada.

```
separa :: Comanda -> [Comanda]
separa Para = [] -- Si para, retorn []
separa (c1 :#: c2) = separa c1 ++ separa c2 -- Si comanda composta,
separa les dues comandes i ajunta les llistes
separa c = [c] -- Si no és comanda composta, retorna la comanda en una
llista
```

A continuació és mostraran els diferents resultats dels jocs de proves preparats per aquest problema, on podem veure que la llista resultant conté les Comandes individuals de la comanda entrada, però sense cap aparició de la Comanda **Para**.

```
Executant joc de prova 1...
Separant (Avança 3 :#: Gira 4 :#: Avança 7 :#: Para):
[Avança 3.0,Gira 4.0,Avança 7.0]
Separant ((Avança 3 :#: Gira 4) :#: (Para :#: Avança 7)):
[Avança 3.0,Gira 4.0,Avança 7.0]
Separant (((Para :#: Avança 3) :#: Gira 4) :#: Avança 7):
[Avança 3.0,Gira 4.0,Avança 7.0]
Separant (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para):
[Avança 5.0,Gira 90.0,Avança 2.0]
Separant (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para):
[Gira 180.0,Avança 10.0,Gira 45.0]
Separant (Avança 3 :#: Avança 4 :#: Gira 90 :#: Gira 180 :#: Para):
[Avança 3.0,Avança 4.0,Gira 90.0,Gira 180.0]
Separant (Avança 1 :#: Para :#: Avança 2 :#: Para :#: Gira 270 :#: Para):
[Avança 1.0,Avança 2.0,Gira 270.0]
Separant (Gira 45 :#: Gira 90 :#: Gira 135 :#: Gira 180 :#: Gira 225 :#: Gira 270 :#: Gira 315 :#: Para):
[Gira 45.0,Gira 90.0,Gira 135.0,Gira 180.0,Gira 225.0,Gira 270.0,Gira 315.0]
```

Problema 2. ajunta

```
ajunta :: [Comanda] -> Comanda
```

Aquesta funció rep una llista de Comandes com a entrada i retorna una única Comanda, possiblement composada dues comandes, i aquestes per dues més, i així successivament fins tenir totes les comandes individuals.

Aquesta funció també té tres casos:

- El primer cas és quan la llista d'entrada és buida. En aquest cas, la funció retorna una Comanda **Para**. Això ens serveix per poder cobrir la possibilitat de que s'entri una llista buida, ja sigui perquè l'usuari així ho ha decidit, o en el cas que *ajunta* rebi la sortida de *separa* i aquesta ha rebut només una comanda **Para**.
- El segon cas és quan la llista d'entrada té només una Comanda individual. En aquest cas, la funció simplement retorna aquesta Comanda.
- El tercer cas és quan la llista d'entrada té dues o més Comandes. En aquest cas, la funció combina la primera Comanda amb la resta utilitzant l'operador **:#:** i crida recursivament a si mateixa amb les Comandes restants fins que només queda una Comanda.

En general, la funció ajunta s'utilitza per combinar una llista de Comandes en una única Comanda. Si la llista d'entrada és buida, la funció retorna una Comanda Para; en cas contrari, retorna una Comanda formada per les Comandes que puguin haver-hi a la llista.

```
ajunta :: [Comanda] -> Comanda
ajunta [] = Para   -- Si la llista és buida, retorna Para
ajunta [c] = c -- Si la llista té un element, retorna l'element
ajunta (c1:c2:cs) = c1 :#: ajunta (c2:cs) -- Si la llista té més d'un
element, ajunta els dos primers i crida recursivament amb la resta
```

Els diferents resultats, fent servir el segon joc de proves es mostraran a continuació, on a diferència del problema anterior, rebem una llista de Comandes i retornem una sola Comanda amb totes les Comandes individuals de la llista:

```
Executant joc de prova 2...
Ajuntant [Avança 3, Gira 4, Avança 7]:
Avança 3.0 :#: Gira 4.0 :#: Avança 7.0
Ajuntant [Avança 5, Gira 90, Avança 2, Para]:
Avança 5.0 :#: Gira 90.0 :#: Avança 2.0 :#: Para
Ajuntant [Gira 180, Avança 10, Gira 45, Para]:
Gira 180.0 :#: Avança 10.0 :#: Gira 45.0 :#: Para
Ajuntant [Avança 3, Avança 4, Gira 90, Gira 180, Para]:
Avança 3.0 :#: Avança 4.0 :#: Gira 90.0 :#: Gira 180.0 :#: Para
Ajuntant [Avança 1, Para, Avança 2, Para, Gira 270, Para]:
Avança 1.0 :#: Para :#: Avança 2.0 :#: Para :#: Gira 270.0 :#: Para
Ajuntant [Gira 45, Gira 90, Gira 135, Gira 180, Gira 225, Gira 270, Gira 315, Para]:
Gira 45.0 :#: Gira 90.0 :#: Gira 135.0 :#: Gira 180.0 :#: Gira 225.0 :#: Gira 270.0 :#: Gira 315.0 :#: Para
```

Problema 3. prop_equivalent

```
prop_equivalent :: Comanda -> Comanda -> Bool
```

Aquesta funció ens retorna True o False depenent si les dues Comandes entrades són equivalents mirant si les dues llistes de comandes originades per la funció separa són equivalents comparant cada comanda amb la de l'altra llista, i si totes dues llistes són iguals, això vol dir que les dues comandes són equivalents. Per comparar dues Comandes, haurem de definir com comparar-les.

La línia `instance Eq Comanda where` defineix una instància de la classe de tipus Eq per al tipus Comanda. Això ens permet utilitzar l'operador **==** per comparar dues Comandes per a la igualtat. Les comandes **Avança**, **Gira**, **Para**, **CanviaColor** i **Branca** es comparen en funció dels seus valors (si és que en tenen). Si dues Comandes del mateix tipus (per exemple, **Gira** i **Gira**) tenen els mateixos valors, són considerades iguals. Si dues Comandes són de tipus diferents (per exemple, **Gira** i **Avança**), no són considerades iguals independentment del seus valors .

La línia `(c11 :#: c12) == (c21 :#: c22)` compara dues Comandes que estan compostes per dues Comandes utilitzant l'operador **:#:**. Si les dues Comandes tenen la mateixa estructura i els seus components són iguals, són considerades iguals. Si dues Comandes tenen estructures diferents o els seus components no són iguals, no són considerades iguals.

En general, prop_equivalent ens diu si dues Comandes són iguals comparant les Comandes individuals que les formen.

```
prop_equivalent :: Comanda -> Comanda -> Bool
prop_equivalent c1 c2 = separa c1 == separa c2 -- Si les llistes de
comandes són iguals, són equivalents
instance Eq Comanda where -- Per poder comparar dues comandes (==)
  Avança n1 == Avança n2 = n1 == n2 -- Si són Avança, compara els
nombres
  Gira n1 == Gira n2 = n1 == n2 -- Si són Gira, compara els nombres
  Para == Para = True -- Si són Para, són iguals
  CanviaColor c1 == CanviaColor c2 = c1 == c2  -- Si són CanviaColor,
compara els colors
  Branca c1 == Branca c2 = c1 == c2 -- Si són Branca, compara les
comandes
  (c11 :#: c12) == (c21 :#: c22) = c11 == c21 && c12 == c22 -- Si són
comandes compostes, compara les comandes que les formen (recursivament)
  _ == _ = False -- Si no són del mateix tipus, no són iguals
```

Resultats dels jocs de proves:

```
Executant joc de proves 3...
Comprovant prop_equivalent (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para) (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para):
True
Comprovant prop_equivalent (Avança 3 :#: Gira 90 :#: Avança 2 :#: Para) (Gira 90 :#: Avança 2 :#: Para):
False
Comprovant prop_equivalent (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para) (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para):
True
Comprovant prop_equivalent (Avança 1 :#: Para) (Avança 1 :#: Avança 1 :#: Para):
False
Comprovant prop_equivalent (Gira 90 :#: Gira 90 :#: Gira 90 :#: Para) (Gira 270 :#: Para):
False
```

En aquest problema també s'han implementat les funcions prop_separa_ajunta i prop_separa:

- prop_separa_ajunta: rep una Comanda per paràmetre, i si comprova si la Comanda resultant de fer separa i després ajunta a la Comanda entrada és equivalent a la passada per paràmetre utilitzant la funció que hem implementat anteriorment, prop_equivalent.

```haskell
prop_separa_ajunta :: Comanda -> Bool
prop_separa_ajunta c = prop_equivalent(ajunta(separa c)) c
```

  En resum, comprovem si separa i ajunta ens generen una comanda equivalent a la original.

  A continuació veiem els jocs de proves d'aquesta funció, i com es pot comprovar, funciona correctament:

```
Executant joc de proves 3.2...
Comprovant prop_separa_ajunta (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para):
True
Comprovant prop_separa_ajunta (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para):
True
Comprovant prop_separa_ajunta (Avança 3 :#: Gira 90 :#: Avança 2 :#: Para):
True
Comprovant prop_separa_ajunta (Avança 1 :#: Para :#: Avança 2 :#: Para :#: Gira 270 :#: Para):
True
Comprovant prop_separa_ajunta (Gira 45 :#: Gira 90 :#: Gira 135 :#: Gira 180 :#: Gira 225 :#: Gira 270 :#: Gira 315 :#: Para):
True
```

- prop_separa: rep una Comanda i serveix per comprovar que la llista resultant per la funció separa no conté cap **Para** ni Comanda composta (**Comanda :#: Comanda**). Per fer això farem servir la funció ja predefinida all, la qual retorna True si tots els elements d'una llista compleixen una condició, en aquest cas, ni_para_ni_compost, on si qualsevol de les comandes que rep és un **Para** o una funció composta retorna False; en cas contrari retorna True.

```haskell
prop_separa :: Comanda -> Bool
prop_separa c = all ni_para_ni_compost (separa c) -- Comprova que totes
les comandes de la llista compleixen la propietat niParaNiCompost
  where
    ni_para_ni_compost Para      = False -- Si és Para, retorna False
    ni_para_ni_compost (_ :#: _) = False -- Si és comanda composta,
retorna False
```

```
    ni_para_ni_compost _          = True -- Si no és Para ni comanda
composta, retorna True
```

En resum, comprovem que separa no ens mantingui cap Comanda **Para** ni Comanda composta a la llista resultant.

Els resultats dels jocs de prova es mostren a continuació:

```
Executant joc de proves 3.3...
Comprovant prop_separa (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para):
Separa:
[Avança 5.0,Gira 90.0,Avança 2.0]
Prop_separa:
True
Comprovant prop_separa (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para):
Separa:
[Gira 180.0,Avança 10.0,Gira 45.0]
Prop_separa:
True
Comprovant prop_separa (Avança 3 :#: Gira 90 :#: Avança 2 :#: Para):
Separa:
[Avança 3.0,Gira 90.0,Avança 2.0]
Prop_separa:
True
Comprovant prop_separa (Avança 1 :#: Para :#: Avança 2 :#: Para :#: Gira 270 :#: Para):
Separa:
[Avança 1.0,Avança 2.0,Gira 270.0]
Prop_separa:
True
Comprovant prop_separa (Gira 45 :#: Gira 90 :#: Gira 135 :#: Gira 180 :#: Gira 225 :#: Gira 270 :#: Gira 315 :#: Para):
Separa:
[Gira 45.0,Gira 90.0,Gira 135.0,Gira 180.0,Gira 225.0,Gira 270.0,Gira 315.0]
Prop_separa:
True
Comprovant prop_separa (Avança 5 :#: Gira 90 :#: Avança 2):
Separa:
[Avança 5.0,Gira 90.0,Avança 2.0]
Prop_separa:
True
Comprovant prop_separa (Gira 180 :#: Avança 10 :#: Gira 45):
Separa:
[Gira 180.0,Avança 10.0,Gira 45.0]
Prop_separa:
True
Comprovant prop_separa (Avança 3 :#: Gira 90 :#: Avança 2):
Separa:
[Avança 3.0,Gira 90.0,Avança 2.0]
Prop_separa:
True
Comprovant prop_separa (Avança 1 :#: Avança 2 :#: Gira 270):
Separa:
[Avança 1.0,Avança 2.0,Gira 270.0]
Prop_separa:
True
Comprovant prop_separa (Gira 45 :#: Gira 90 :#: Gira 135 :#: Gira 180 :#: Gira 225 :#: Gira 270 :#: Gira 315):
Separa:
[Gira 45.0,Gira 90.0,Gira 135.0,Gira 180.0,Gira 225.0,Gira 270.0,Gira 315.0]
Prop_separa:
True
```

Problema 4. copia

```
copia :: Int -> Comanda -> Comanda
```

La funció copia ens permetrà copiar una Comanda ‘c’ un nombre ‘n’ de vegades (tant la Comanda com el nombre seran passats per paràmetre), retornant una Comanda composta resultant d’aquesta repetició.

Aquesta funció té dos casos:

- Quan n és igual a 1, en aquest cas, es retorna la Comanda entrada
- Quan n és superior a 1, es crea una Comanda composta utilitzant l’operador **:#:** amb la comanda entrada i el resultat de `copia (n-1) c`.

Per tant, fem servir la recursivitat per anar generant comandes compostes fins que n té valor 1, que en aquest moment retornem ‘c’, donant com a la resultat la Comanda que retornarem.

```
copia :: Int -> Comanda -> Comanda
copia 1 c = c -- Si n = 1, retorna la comanda
copia n c = c :#: copia (n - 1) c -- Si n > 1, retorna la comanda i
crida recursivament amb n-1
```

Aquí mostrem els diferents resultats amb la funció copia:

```
Executant joc de proves 4...
Comprovant copia 3 (Avança 10 :#: Gira 120):
Avança 10.0 :#: Gira 120.0 :#: Avança 10.0 :#: Gira 120.0 :#: Avança 10.0 :#: Gira 120.0
Comprovant copia 3 (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para):
Avança 5.0 :#: Gira 90.0 :#: Avança 2.0 :#: Para :#: Avança 5.0 :#: Gira 90.0 :#: Avança 2.0 :#: Para :#: Avança 5.0 :#: Gira 90.0 :#: Ava
nça 2.0 :#: Para
Comprovant copia 2 (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para):
Gira 180.0 :#: Avança 10.0 :#: Gira 45.0 :#: Para :#: Gira 180.0 :#: Avança 10.0 :#: Gira 45.0 :#: Para
Comprovant copia 4 (Avança 3 :#: Gira 90 :#: Avança 2 :#: Para):
Avança 3.0 :#: Gira 90.0 :#: Avança 2.0 :#: Para :#: Avança 3.0 :#: Gira 90.0 :#: Avança 2.0 :#: Para :#: Avança 3.0 :#: Gira 90.0 :#: Ava
nça 2.0 :#: Para :#: Avança 3.0 :#: Gira 90.0 :#: Avança 2.0 :#: Para
Comprovant copia 1 (Avança 1 :#: Para :#: Avança 2 :#: Para :#: Gira 270 :#: Para):
Avança 1.0 :#: Para :#: Avança 2.0 :#: Para :#: Gira 270.0 :#: Para
```

Problema 5. pentagon

```
pentagon :: Distancia -> Comanda
```

Aquest funció ens permetrà generar la Comanda que ens permetran formar un pentagon amb una mida de costat entrada per paràmetre.

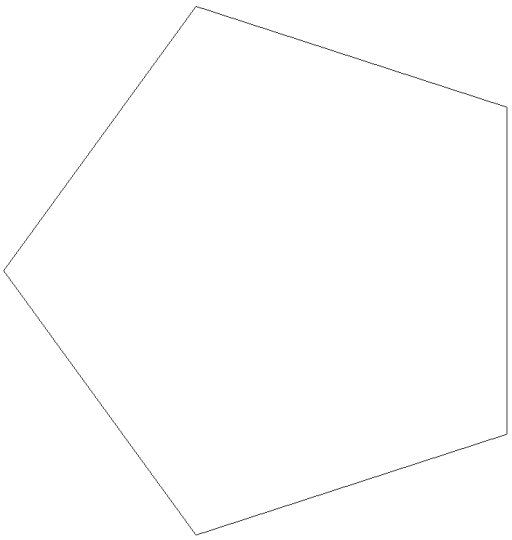
Per generar aquesta Comanda, aprofitarem la funció copia creada anteriorment per crear la comanda que ens servirà per crear aquest pentagon, copiant 5 vegades la Comanda **Avança distancia :#: Gira 72**, on distància serà el valor entrat. D'aquesta manera, s'avançarà la distància especificada i es giraran els 72º corresponents a cada angle d'un pentàgon regular.

```
pentagon :: Distancia -> Comanda
pentagon distancia = copia 5 (Avança distancia :#: Gira 72)
```

Aquí podem veure els diferents resultats depenent de quin valor es passi per paràmetre:

```
Executant joc de proves 5...
Comprovant pentagon 10:
Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0
Comprovant pentagon 5:
Avança 5.0 :#: Gira 72.0 :#: Avança 5.0 :#: Gira 72.0 :#: Avança 5.0 :#: Gira 72.0 :#: Avança 5.0 :#: Gira 72.0 :#: Avança 5.0 :#: Gira 72.0
Comprovant pentagon 15:
Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0
Comprovant pentagon 8:
Avança 8.0 :#: Gira 72.0 :#: Avança 8.0 :#: Gira 72.0 :#: Avança 8.0 :#: Gira 72.0 :#: Avança 8.0 :#: Gira 72.0 :#: Avança 8.0 :#: Gira 72.0
Comprovant pentagon 20:
Avança 20.0 :#: Gira 72.0 :#: Avança 20.0 :#: Gira 72.0 :#: Avança 20.0 :#: Gira 72.0 :#: Avança 20.0 :#: Gira 72.0 :#: Avança 20.0 :#: Gira 72.0
```

Aquí mostrarem el resultat de fer display (pentagon 10):



Problema 6. poligon

```
poligon :: Distancia -> Int -> Angle -> Comanda
```

Amb aquesta funció, el que farem serà crear una Comanda per crear un polígon regular a partir de les dades entrades, en aquest cas, la distància que tindrà cada costat, el nombre de costats, i l'angle entre costats.

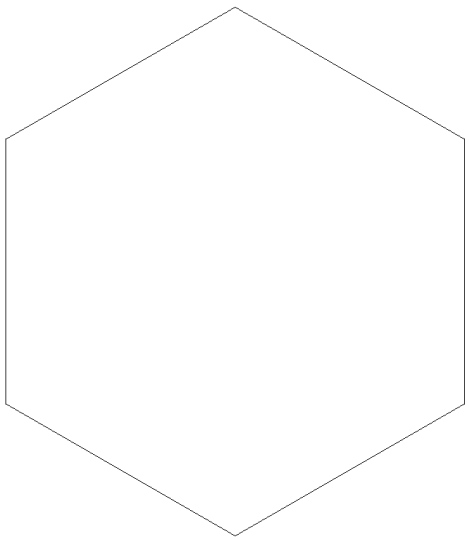
Igual que a la funció anterior, aprofitarem la funció copia per generar la nostra Comanda, fent tantes còpies de la Comanda com costats que tingui la figura que volem crear.

```
poligon :: Distancia -> Int -> Angle -> Comanda
poligon distancia nCostats angle = copia nCostats (Avança distancia :#:
Gira angle) -- Crida a copia per copiar nCostats vegades la comanda
(Avança distancia :#: Gira angle)
```

A continuació mostrem els diferents resultats amb el joc de proves:

```
Executant joc de proves 6.1...
Comprovant poligon 10 5 72:
Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gi
ra 72.0
Comprovant poligon 10 3 120:
Avança 10.0 :#: Gira 120.0 :#: Avança 10.0 :#: Gira 120.0 :#: Avança 10.0 :#: Gira 120.0
Comprovant poligon 5 4 90:
Avança 5.0 :#: Gira 90.0 :#: Avança 5.0 :#: Gira 90.0 :#: Avança 5.0 :#: Gira 90.0 :#: Avança 5.0 :#: Gira 90.0
Comprovant poligon 15 5 72:
Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gira 72.0 :#: Avança 15.0 :#: Gi
ra 72.0
Comprovant poligon 8 6 60:
Avança 8.0 :#: Gira 60.0 :#: Avança 8.0 :#: Gira 60.0 :#: Avança 8.0 :#: Gira 60.0 :#: Avança 8.0 :#: Gira 60.0 :#: Avança 8.0 :#: Gira 60
.0 :#: Avança 8.0 :#: Gira 60.0
Comprovant poligon 20 8 45:
Avança 20.0 :#: Gira 45.0 :#: Avança 20.0 :#: Gira 45.0 :#: Avança 20.0 :#: Gira 45.0 :#: Avança 20.0 :#: Gira 45.0 :#: Avança 20.0 :#: Gi
ra 45.0 :#: Avança 20.0 :#: Gira 45.0 :#: Avança 20.0 :#: Gira 45.0 :#: Avança 20.0 :#: Gira 45.0
```

També mostrarem el resultat de fer 3- display (poligon 8 6 60):



Per aquest problema, també se'ns ha demanat fer una funció que ens digui si podem generar una Comanda igual a la que generaríem si volguéssim fer un pentàgon.

Per fer això, aprofitarem les funcions prop_equivalent i pentagon, fent que passada una distància per paràmetre, mirem si acabem generant una Comanda equivalent tant amb el mètode poligon com pentagon.

```
prop_poligon_pentagon :: Distancia -> Bool
prop_poligon_pentagon distancia = prop_equivalent (poligon distancia 5
72) (pentagon distancia)
```

A continuació mostrem els resultats de les diferents execucions:

```
Executant joc de proves 6.2...
Comprovant prop_poligon_pentagon 5:
True
Comprovant prop_poligon_pentagon 10:
True
Comprovant prop_poligon_pentagon 15:
True
Comprovant prop_poligon_pentagon 8:
True
Comprovant prop_poligon_pentagon 20:
True
Realitzant comprovació amb quickCheck prop_poligon_pentagon:
+++ OK, passed 100 tests.
```

Problema 7. espiral

```
espiral :: Distancia -> Int -> Distancia -> Angle -> Comanda
```

La funció espiral és una funció que retorna una comanda per dibuixar una espiral. La comanda es crea utilitzant la funció auxiliar pas_espiral, que repeteix una comanda 'nSegm' vegades. La comanda que es repeteix és **Avança costat :#: Gira angle**, que mou el llapis endavant 'costat' unitats i gira 'angle' graus a l'esquerra. A cada iteració, la longitud del costat s'incrementa en pas (el qual pot ser negatiu). Així, la funció espiral crea una comanda que dibuixa una espiral amb nSegm segments, cada un amb una longitud de costat inicial de costat i incrementant en pas a cada iteració, i amb un angle de gir de angle graus.

La funció pas_espiral és una funció auxiliar que repeteix una comanda 'nSegm' vegades. Si la longitud del costat és negativa o nSegm és 0, la funció retorna **Para**. Si la longitud del costat és positiva i nSegm és més gran que 0, la funció retorna `Avança costat :#: Gira angle :#: pas_espiral (costat+pas) (nSegm-1)`, que mou el llapis endavant costat unitats, gira 'angle' graus a l'esquerra, i crida recursivament la mateixa funció amb una longitud de costat incrementada en pas i 'nSegm' decrementat en 1.

```
espiral :: Distancia -> Int -> Distancia -> Angle -> Comanda
espiral costat nSegm pas angle = pas_espiral costat nSegm
  where
  pas_espiral costat nSegm | costat <= 0 || nSegm == 0 = Para -- Si la
longitud del costat és negativa o nSegm = 0, retorna Para
```

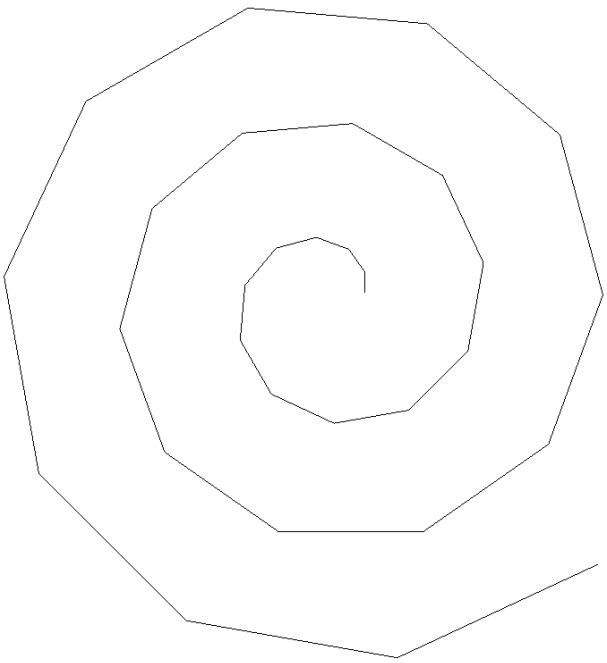
```
        | otherwise = Avança costat :#: Gira angle :#: pas_espiral
(costat+pas) (nSegm-1)-- Si la longitud del costat és positiva i nSegm >
0, retorna Avança costat :#: Gira angle :#: pas_espiral (costat+pas)
(nSegm-1)
```

En resum, es tracta d'una funció que per generar la comanda crida recursivament una funció auxiliar que rep i actualitza la nova distància del costat en cada iteració.

Aquí tenim la comanda resultant dels diferents jocs de prova:

```
Executant joc de proves 7.2...
Comprovant espiral 30 5 10 35:
Avança 30.0 :#: Gira 35.0 :#: Avança 40.0 :#: Gira 35.0 :#: Avança 50.0 :#: Gira 35.0 :#: Avança 60.0 :#: Gira 35.0 :#: Avança 70.0 :#: Gi
ra 35.0 :#: Para
Comprovant espiral 10 3 5 90:
Avança 10.0 :#: Gira 90.0 :#: Avança 15.0 :#: Gira 90.0 :#: Avança 20.0 :#: Gira 90.0 :#: Para
Comprovant espiral (-5) 4 2 45:
Para
Comprovant espiral 15 5 (-3) 60:
Avança 15.0 :#: Gira 60.0 :#: Avança 12.0 :#: Gira 60.0 :#: Avança 9.0 :#: Gira 60.0 :#: Avança 6.0 :#: Gira 60.0 :#: Avança 3.0 :#: Gira
60.0 :#: Para
Comprovant espiral 8 6 0 30:
Avança 8.0 :#: Gira 30.0 :#: Avança 8.0 :#: Gira 30.0 :#: Avança 8.0 :#: Gira 30.0 :#: Avança 8.0 :#: Gira 30.0 :#: Avança 8.0 :#: Gira 30
.0 :#: Avança 8.0 :#: Gira 30.0 :#: Para
Comprovant espiral 20 8 (-10) 72:
Avança 20.0 :#: Gira 72.0 :#: Avança 10.0 :#: Gira 72.0 :#: Para
Comprovant espiral 30 2 (-10) 35:
Avança 30.0 :#: Gira 35.0 :#: Avança 20.0 :#: Gira 35.0 :#: Para
```

I també mostrarem el resultat de fer display (espiral 30 30 10 35):



Problema 8. execute

```
execute :: Comanda -> [Ln]
```

La funció execute rep una comanda i retorna una llista de línies a dibuixar i la posició final de la tortuga. La tortuga és una tupla que conté el llapis actual, l'angle de la tortuga i la posició actual de la tortuga.

La funció processa és una funció auxiliar que rep una comanda i una Turtle, i retorna una llista de línies a dibuixar i la nova posició de la Turtle. Si la comanda és una seqüència de comandes **c :#: d**, la funció crida recursivament la mateixa funció amb la comanda **c** i la Turtle original, i després crida la mateixa funció amb la comanda **d** i la nova Turtle. Això permet processar les comandes en ordre. Si la comanda és una branca **Branca c**, la funció crida recursivament la mateixa funció amb la comanda **c** i la Turtle original, però no actualitza la Turtle, permetent tornar a ons trobàvem abans de la comanda **Branca**. Si la comanda és **Avança dist**, la funció actualitza la llista de línies amb una nova línia que va des del punt actual de la Turtle fins al nou punt després de moure la distància 'dist' en la direcció de l'angle actual de la Turtle. La funció també actualitza la posició de la Turtle. Si la comanda és **Gira delta**, la funció actualitza l'angle de la Turtle. Si la comanda és **CanviaColor nou**, la funció actualitza el llapis de la Turtle. Si la comanda és **Para**, la funció no fa res.

```
execute :: Comanda -> [Ln]
execute c = linies
  where
    (linies, turtle) = processa c (negre, 0, Pnt 0 0) -- La tortuga
comença en el punt (0,0) i mirant cap a l'eix positiu de les X

    -- DESCRIPCIÓ: Donada una comanda i una Turtle, retorna una llista
de línies a pintar i la posició final de la Turtle
    processa :: Comanda -> Turtle -> ([Ln], Turtle)
    processa (c :#: d) turtle = (liniesC ++ liniesD, turtleD)
      where --Concatenem el que hem de dibuixar de C i de D, i per la
Turtle, fem la C i amb el que ens doni el C, fem el D
        (liniesC, turtleC) = processa c turtle
        (liniesD, turtleD) = processa d turtleC
    -- Si és una branca, actualitzem el dibuix però no actualitzem el
Turtle (rama principal)
    processa (Branca c) turtle = (liniesC, turtle)
      where (liniesC, turtleC) = processa c turtle
    -- Si avancem --> Actualitzem [Ln] amb el nou dibuix (fa un dibuix
amb el llapis de punt que és puntAntic a puntFinal), actualitzem la
posició final del Turtle
    processa (Avança dist) (llapis, angle, punt) = ([Ln llapis punt
puntFinal | llapis /= Transparent], (llapis, angle, puntFinal))
      where puntFinal = punt + scalar dist * converteix_polar angle
    processa (Gira delta) (llapis, angle, punt) = ([], (llapis, angle-
```

```
delta, punt)) -- No fem res a la llista Ln, només actualitzem l'angle
(Turtle)
    processa (CanviaColor nou) (vell, angle, punt) = ([], (nou, angle,
punt)) -- No fem res a la llista Ln, només actualitzem el llapis
(Turtle)
    processa Para turtle = ([], turtle) -- No fem res
```

La funció converteix_polar és una funció auxiliar que rep un angle en graus i retorna un punt que representa el vector unitari que apunta en la direcció de l'angle.

```
converteix_polar :: Angle -> Pnt
converteix_polar angle = Pnt (cos radiants) (sin radiants)
  where radiants = angle * 2 * pi / 360
```

A continuació mostrem diverses execucions de la funció execute:

```
Executant joc de proves 8...
Executant execute (Avança 30 :#: Para :#: Gira 10 :#: Avança 20):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 30.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 30.0 0.0) (Pnt 49.696156 (-3.4729638))]
Executant execute (Avança 30 :#: Para :#: Gira 10 :#: Avança 20 :#: Gira (-15) :#: Para :#: Avança 10 :#: Para :#: Para):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 30.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 30.0 0.0) (Pnt 49.696156 (-3.4729638)),Ln (Color' 0.0 0
.0 0.0) (Pnt 49.696156 (-3.4729638)) (Pnt 59.658104 (-2.6014063))]
Executant execute (Avança 50 :#: Gira 90 :#: Avança 30 :#: Gira 45 :#: Avança 20 :#: Para):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 50.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 50.0 0.0) (Pnt 50.0 (-30.0)),Ln (Color' 0.0 0.0 0.0) (P
nt 50.0 (-30.0)) (Pnt 35.857864 (-44.142136))]
Executant execute (Avança 10 :#: Gira 180 :#: Avança 20 :#: Gira 45 :#: Para):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 10.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 10.0 0.0) (Pnt (-10.0) 1.7484556e-6)]
Executant execute (Gira 45 :#: Avança 30 :#: Gira 90 :#: Para):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 21.213203 (-21.213203))]
Executant execute (Branca (Avança 20 :#: Gira 90) :#: Gira 45 :#: Avança 30 :#: Para):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 20.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 21.213203 (-21.213203))]
Executant execute (Avança 50 :#: Branca (Gira 180 :#: Avança 20) :#: Avança 30 :#: Gira 45 :#: Para):
[Ln (Color' 0.0 0.0 0.0) (Pnt 0.0 0.0) (Pnt 50.0 0.0),Ln (Color' 0.0 0.0 0.0) (Pnt 50.0 0.0) (Pnt 30.0 1.7484556e-6),Ln (Color' 0.0 0.0 0.
0) (Pnt 50.0 0.0) (Pnt 80.0 0.0)]
```

Problema 9. optimitza

```
optimitza :: Comanda -> Comanda
```

La funció optimitza rep una Comanda i retorna una Comanda equivalent però optimitzada; és a dir, no hi ha cap comanda **Para**, ni **Avança 0**, ni **Gira 0**, i no hi ha dos o més **Avança** consecutius (el mateix amb els **Gira**).

La funció optimitza genera la comanda seguint una sèrie de passos. Primer, separa la comanda en una llista de comandes individuals amb la funció separa (d'aquesta manera eliminem també els **Para**). Després, filtra la llista per eliminar les comandes **Avança 0** (considerarem que mai es tindrà cap **Avança** que tingui un valor negatiu com a resultat de combinar, per tant, no caldrà filtrar-los més). A continuació, combinem (amb l'ajut del mètode auxiliar combinar) les comandes **Gira** seguides i les comandes **Avança** seguides en una sola comanda. Una vegada més, filtrem la llista, eliminant tots els possibles **Gira 0** que poguessin haver aparegut durant la combinació. Després, fem servir una vegada més el mètode auxiliar combinar per poder combinar els **Avança** consecutius una vegada més (ja que al filtrar els **Gira 0**, existeix de nou la possibilitat que hi hagi **Avança** consecutius).

La funció combinar és una funció auxiliar que rep una llista de comandes i les combina segons les regles especificades. Si la llista està buida, la funció retorna una llista buida. Si la llista conté dues comandes **Gira** seguides, la funció les combina en una sola comanda **Gira** amb la suma dels angles. Si la llista conté dues comandes **Avança** seguides, la funció les combina en una sola comanda **Avança** amb la suma de les distàncies. Si la llista conté una sola comanda, la funció retorna la llista sense canvis. Amb aquestes regles, anirem recorrent la llista de forma recursiva fins llegir-la completament, combinant totes les Comandes que hagin sigut possibles.

```
optimitza :: Comanda -> Comanda
-- Primer separem, filtrem per avança, combinem, filtrem per fira,
combinem i ajuntem
optimitza = ajunta . combinar . filter (/= Gira 0) . combinar . filter
(/= Avança 0) . separa
    where
      combinar [] = []
      combinar (Gira x : Gira y : xs) = combinar (Gira (x+y) : xs) -- Si
hi ha dues comandes Gira seguides, les combina
      combinar (Avança x : Avança y : xs) = combinar (Avança (x+y) : xs)
-- Si hi ha dues comandes Avança seguides, les combina
      combinar (x:xs) = x : combinar xs -- Si no hi ha dues comandes
seguides, crida recursivament amb la resta de la llista
```

En resum, optimitza és una funció que modifica la Comanda rebuda de tal manera que s'eliminin les Comandes que no afecten al dibuix a l'hora de fer-ho, i combina els **Avança** consecutius i els **Gira** consecutius per tal de reduir el nombre de Comandes individuals.

A continuació tenim els diferents jocs de proves, i com es pot apreciar, l'optimització sembla funcionar correctament:

```
Executant joc de proves 9...
Executant optimitza (Avança 10 :#: Para :#: Avança 20 :#: Gira 35 :#: Avança 0 :#: Gira 15 :#: Gira (-50)):
Avança 30.0
Executant optimitza (Avança 10 :#: Para :#: Gira 40 :#: Avança 20 :#: Gira (-40)):
Avança 10.0 :#: Gira 40.0 :#: Avança 20.0 :#: Gira -40.0
Executant optimitza (Avança 10 :#: Para :#: Gira 40 :#: Gira (-30) :#: Avança 20):
Avança 10.0 :#: Gira 10.0 :#: Avança 20.0
Executant optimitza (Avança 10 :#: Para :#: Avança (-10)):
Avança 0.0
Executant optimitza (Avança 5 :#: Gira 90 :#: Avança 2 :#: Para):
Avança 5.0 :#: Gira 90.0 :#: Avança 2.0
Executant optimitza (Gira 180 :#: Avança 10 :#: Gira 45 :#: Para):
Gira 180.0 :#: Avança 10.0 :#: Gira 45.0
Executant optimitza (Avança 3 :#: Gira 90 :#: Avança 2 :#: Para):
Avança 3.0 :#: Gira 90.0 :#: Avança 2.0
Executant optimitza (Avança 1 :#: Para :#: Avança 2 :#: Para :#: Gira 270 :#: Para):
Avança 3.0 :#: Gira 270.0
Executant optimitza (Gira 45 :#: Gira 90 :#: Gira 135 :#: Gira 180 :#: Gira 225 :#: Gira 270 :#: Gira 315 :#: Para):
Gira 1260.0
Executant optimitza (Avança 5 :#: Gira 90 :#: Avança 2 :#: Gira 0 :#: Avança 0 :#: Gira 45 :#: Para):
Avança 5.0 :#: Gira 90.0 :#: Avança 2.0 :#: Gira 45.0
Executant optimitza (Gira 180 :#: Avança 10 :#: Gira 0 :#: Gira 0 :#: Avança 0 :#: Gira 45 :#: Para):
Gira 180.0 :#: Avança 10.0 :#: Gira 45.0
Executant optimitza (Avança 3 :#: Gira 90 :#: Avança 0 :#: Avança 0 :#: Gira 0 :#: Gira 0 :#: Avança 2 :#: Para):
Avança 3.0 :#: Gira 90.0 :#: Avança 2.0
Executant optimitza (Avança 1 :#: Para :#: Avança 0 :#: Avança 0 :#: Gira 270 :#: Para):
Avança 1.0 :#: Gira 270.0
Executant optimitza (Gira 45 :#: Gira 90 :#: Avança 0 :#: Gira 0 :#: Avança 0 :#: Gira 315 :#: Gira 0 :#: Gira 0 :#: Para):
Gira 450.0
```

Problema 10. triangle

```
triangle :: Int -> Comanda
```

A partir d'aquí, començarem a fer les funcions que ens retornen Comandes per generar els fractals que se'ns demanin passat per paràmetre un nombre d'iteracions. Per realitzar aquesta tasca i les posteriors, farem servir regles de reescriptura, on a partir d'una situació inicial, anirem reescribint les lletres per un conjunt de signes i lletres a cada iteració (els signes '+' i '-' representaran la Comanda **Gira** en un sentit o l'altre, mentre que les lletres representaran la Comanda **Avança**). Una vegada s'hagin fet les iteracions necessàries, es retornarà una Comanda que es farà servir per dibuixar el fractal en qüestió.
Per exemple, en el cas del triangle d'aquest problema farem servir la següent gramàtica:

```
angle:          90
inici:          +f
reescriptura:   f → f+f-f-f+f
```

On començarem amb "+f", i a cada iteració, substituirem cada 'f' que aparegui per "f+f-f-f+f", i al final es farà la substitució de caràcters per la seva Comanda respectiva.

Nosaltres, el que farem serà canviar una mica la gramàtica per tal de facilitar-nos la feina a l'hora de fer les subtitucions: els signes '+' i '-' els representarem com a "pos" i "neg" respectivament, i quan es facin les subtitucions els substituirem per Comandes **Gira 90** i **Gira (-90)** respectivament (són 90º perquè són els especificats a la gramàtica). En el cas de les lletres, les tractarem com a funcions que reben un número com a paràmetre que representarà quants nivells queden per acabar les iteracions, on mentre el paràmetre rebut sigui superior a 0, es retornarà el resultat de la reescriptura d'aquella lletra i es reduirà en 1 el nivell que passarem per paràmetre a les lletres resultants de la reescriptura. D'aquesta manera, podrem anar reescribint les lletres de forma recursiva, fins arribar al nivell 0, on es substituirà la lletra per una Comanda **Avança**.

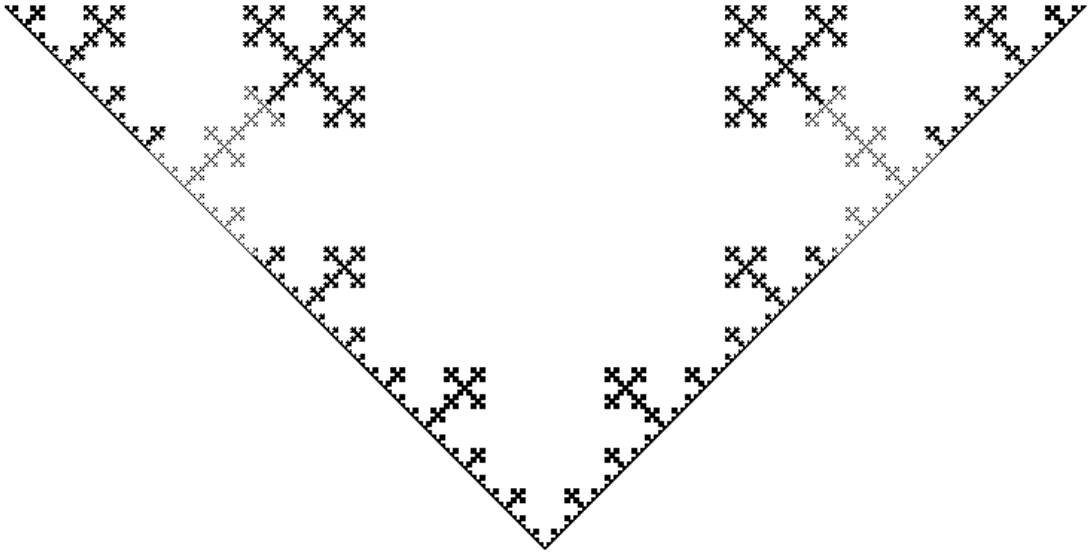
```
triangle :: Int -> Comanda
triangle lvl  =  pos :#: f lvl
  where
    f 0  = Avança 10 -- Si f = 0, retorna Avança 10
    f lvl  = f (lvl-1) :#: pos :#: f (lvl-1) :#: neg :#: f (lvl-1) :#:
neg :#: f (lvl-1) :#: pos :#: f (lvl-1) -- Va fent recursivitat amb les
crides fins que valgui 0
    neg  = Gira (-90) -- Representa signe '-' de la gramàtica
    pos  = Gira 90    -- Representa signe '+' de la gramàtica
```

D'aquesta manera, podrem substituir els caràcters per les Comandes pertinents de forma simple, i ens serveix pels futurs problemes.

A continuació mostrarem els resultats, no només de la Comanda resultant de fer triangle 1, sinó també del dibuix que en genera cridar la funció triangle amb un valor més gran:

NT: els dibuixos que es fan als problemes a continuació són de nivell 7

```
Executant joc de proves 10 (triangle 1)...
Gira 90.0 :#: Avança 10.0 :#: Gira 90.0 :#: Avança 10.0 :#: Gira -90.0 :#: Avança 10.0 :#: Gira -90.0 :#: Avança 10.0 :#: Gira 90.0 :#: Av
ança 10.0
PS D:\Deures\3rGEINF\Paradigmes\haskell\JohnTheArtist> cabal run
```



Problema 11. fulla

```
fulla :: Int -> Comanda
```

Ara introduirem un nou concepte dins la gramàtica, que és el concepte de la branca. A la gramàtica que se'ns proporcionava, representàvem el començament d'una branca amb '[' i la seva finalització amb ']', i una vegada finalitzada la branca, tornarem a l'estat que ens trobàvem abans d'inicialitzar la branca.

Nosaltres representarem la branca fent servir la Comanda **Branca**, la qual rebrà una Comanda com a valor, i una vegada finalitzada aquesta Comanda, es tornarà automàticament a l'estat anterior de d'inicialitzar la Comanda **Branca** (el dibuix realitzat dins la branca sí que es mantindrà). Per tant, tot el que hauria de ser-hi entre claus ([ ]) serà el valor que es passarà a la Comanda **Branca**.

També s'introdueix la Comanda **CanviColor**, la qual fem servir per canviar el color del llapis.

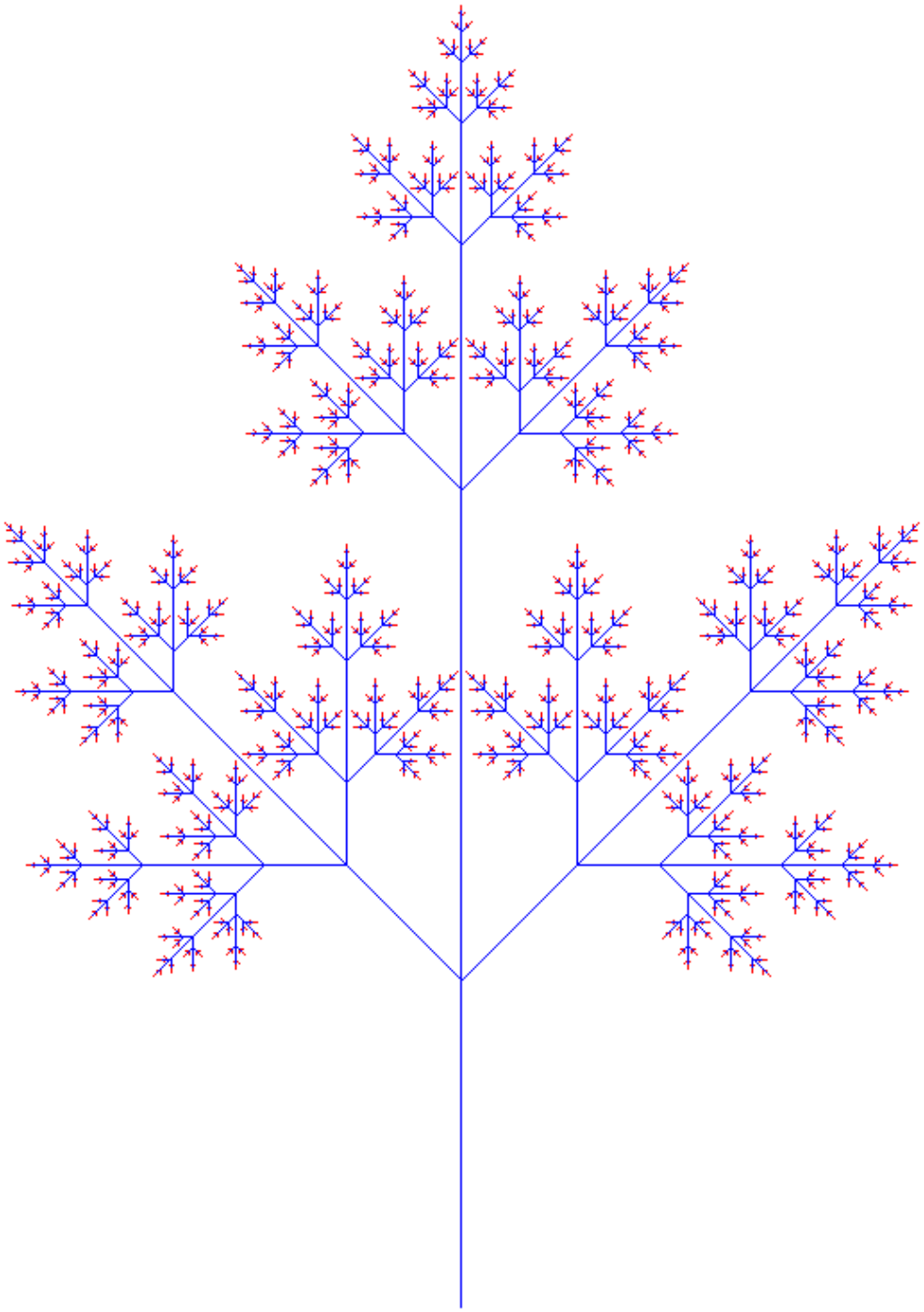
El procés de substitució de caràcters és el mateix que s'ha realitzat en el problema anterior, però amb variacions a la gramàtica:

```
angle:          45
inici:          f
reescriptura:   f → g[-f][+f][gf]
                g → gg
```

```
fulla :: Int -> Comanda
fulla lvl  =  f lvl -- lvl (level o nivell) serà el nombre de vegades
que es farà recursivitat de la gramàtica
  where
    f 0 = CanviaColor vermell :#: Avança 10 -- Si f = 0, retorna Avança
10
    f lvl = g (lvl-1) :#: Branca (neg :#: f (lvl-1)) :#: Branca (pos :#:
f (lvl-1)) :#: Branca (g (lvl-1) :#: f (lvl-1)) -- Va fent recursivitat
amb les crides fins que valgui 0
    g 0 = CanviaColor blau :#: Avança 10 -- Si g = 0, retorna Avança 10
    g lvl = g (lvl-1) :#: g (lvl-1) -- Va fent recursivitat amb les
crides fins que valgui 0
    neg = Gira (-45)  -- Representa signe '-' de la gramàtica
    pos = Gira 45     -- Representa signe '+' de la gramàtica
```

A continuació mostrem la comanda resultant de fer fulla 1, i el resultat visual de fulla amb un valor més gran:

```
11
Executant joc de proves 11 (fulla 1)...
CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: Brancament Gira -45.0 :#: CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Brancament
 Gira 45.0 :#: CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Brancament CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: CanviaColo
r Color' 1.0 0.0 0.0 :#: Avança 10.0
```



Problema 12. hilbert

```
hilbert :: Int -> Comanda
```

A partir d'aquí, farem servir sempre l'estrategia dels dos problemes anteriors per resoldre els problemes.

És a dir:

- Fer reescriptura de les lletres mentre encara quedin iteracions per fer, i una vegada ja no quedin més iteracions a realitzar, canviar les lletres per **Avança**.
- Substituir els "pos" i "neg" per **Gira** amb l'angle corresponent.
- En cas d'haver-hi [ ] a la gramàtica, representar-los amb la Comanda **Branca**.

A continuació mostrem la gramàtica:

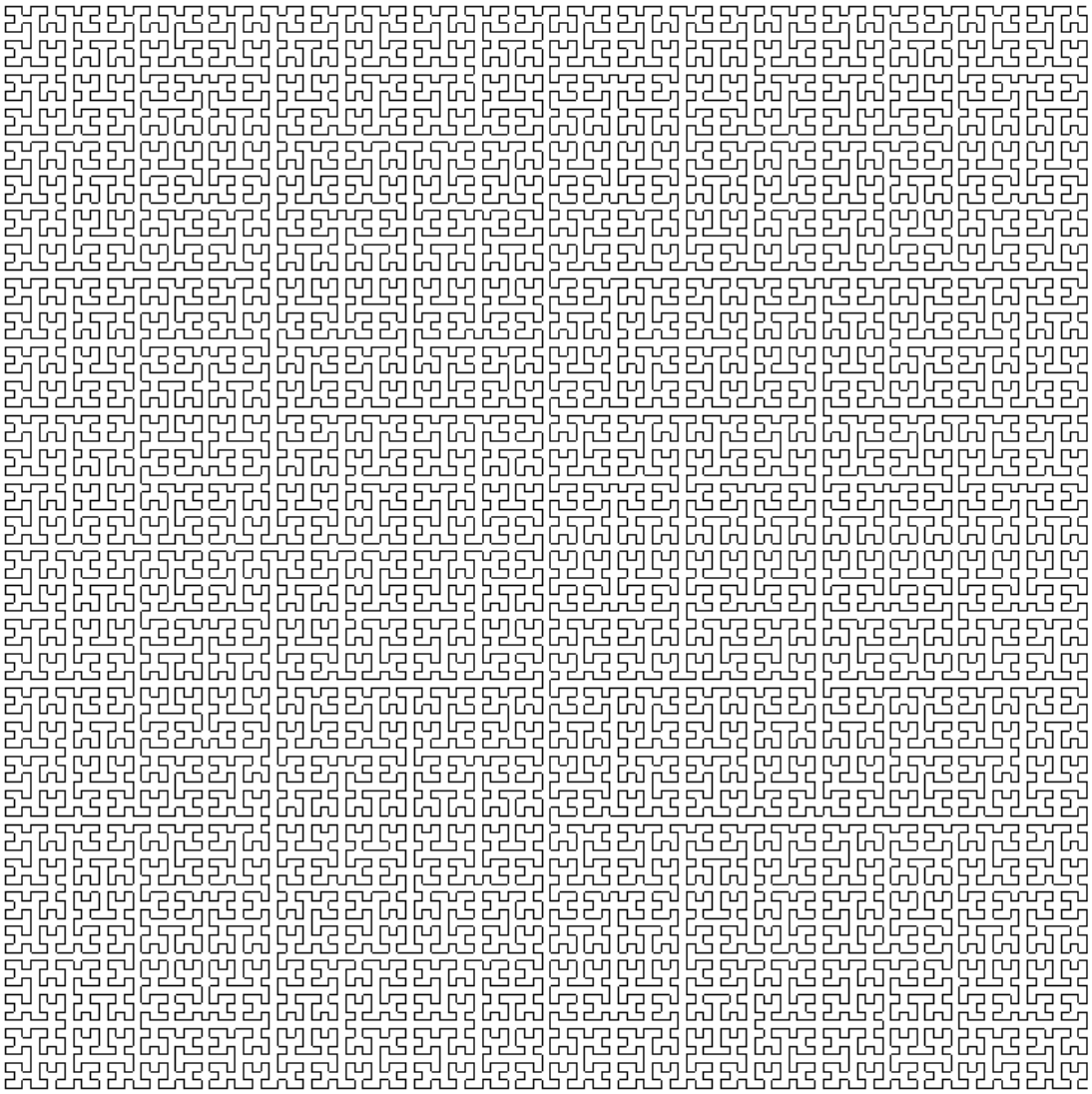
```
angle:          90
inici:          l
reescritura:    l → +rf-lfl-fr+
                r → -lf+rfr+fl-
```

I el codi es mostrarà a continuació:

```
hilbert :: Int -> Comanda
hilbert lvl = l lvl
  where
    l 0 = Para -- Si lvl = 0, retorna Para
    l lvl = pos :#: r (lvl-1) :#: f :#: neg :#: l (lvl-1) :#: f :#: l
(lvl-1) :#: neg :#: f :#: r (lvl-1) :#: pos -- Va fent recursivitat amb
les crides fins que valgui 0
    r 0 = Para -- Si lvl = 0, retorna Para
    r lvl = neg :#: l (lvl-1) :#: f :#: pos :#: r (lvl-1) :#: f :#: r
(lvl-1) :#: pos :#: f :#: l (lvl-1) :#: neg -- Va fent recursivitat amb
les crides fins que valgui 0
    f = Avança 10 -- Si f = 0, retorna Avança 10
    neg = Gira (-90)  -- Representa signe '-' de la gramàtica
    pos = Gira 90     -- Representa signe '+' de la gramàtica
```

A continuació mostrem els diferents resultats:

```
12
Executant joc de proves 12 (hilbert 1)...
Gira 90.0 :#: Para :#: Avança 10.0 :#: Gira -90.0 :#: Para :#: Avança 10.0 :#: Para :#: Gira -90.0 :#: Avança 10.0 :#: Para :#: Gira 90.0
```



Problema 13. fletxa

```
fletxa :: Int -> Comanda
```

Aquesta serà la gramàtica que es farà servir per aquest problema:

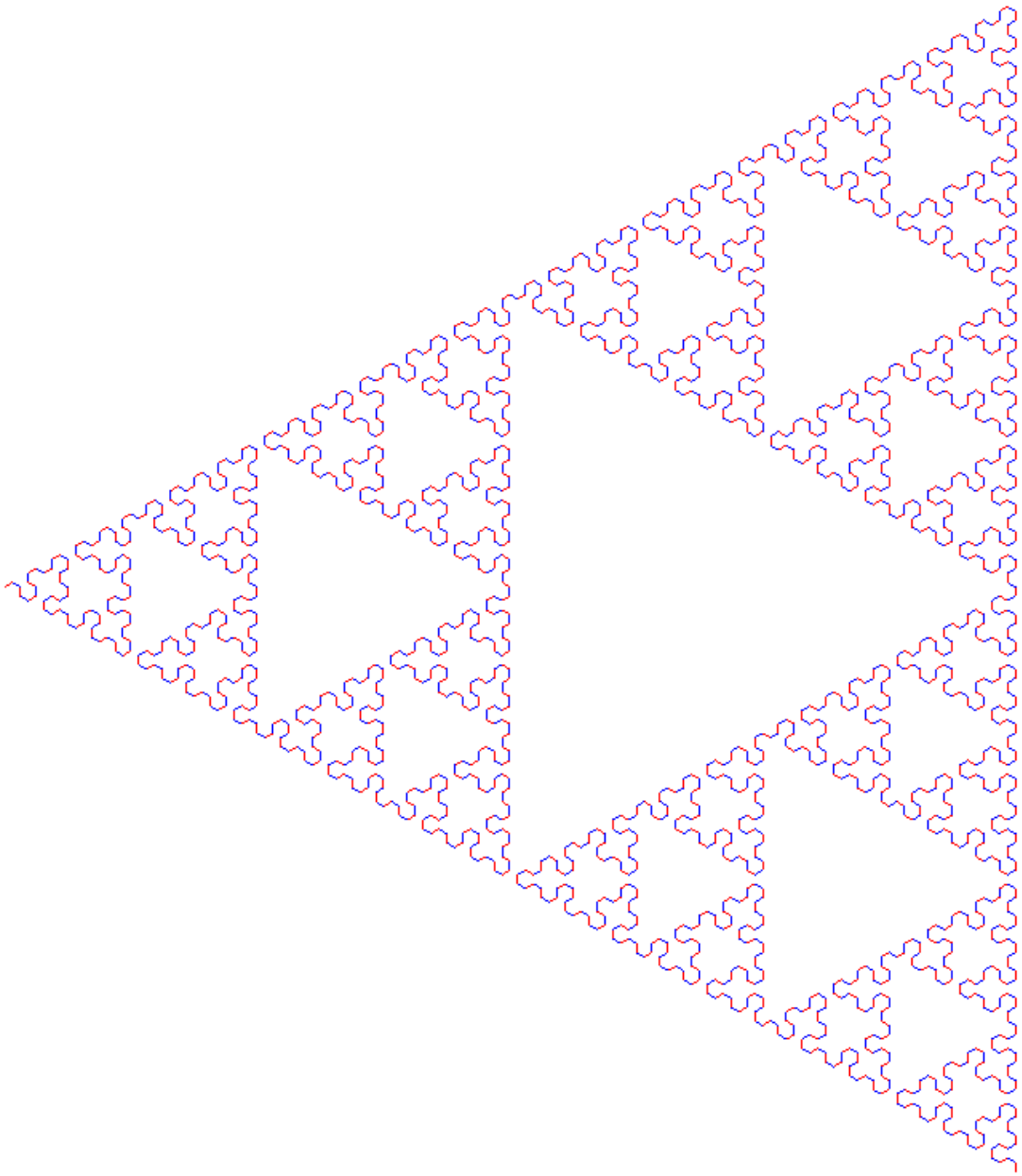
```
angle:          60
inici:          f
reescritura:    f → g+f+g
                g → f-g-f
```

El codi es mostrarà a continuació:

```
fletxa :: Int -> Comanda
fletxa lvl = f lvl
  where
    f 0 = CanviaColor blau :#: Avança 10 -- Si lvl = 0, retorna Avança
10
    f lvl = g (lvl-1) :#: neg :#: f (lvl-1) :#: neg :#: g (lvl-1) -- Va
fent recursivitat amb les crides fins que valgui 0
    g 0 = CanviaColor vermell :#: Avança 10 -- Si lvl = 0, retorna
Avança 10
    g lvl = f (lvl-1) :#: pos :#: g (lvl-1) :#: pos :#: f (lvl-1) -- Va
fent recursivitat amb les crides fins que valgui 0
    neg = Gira 60 -- Representa signe '-' de la gramàtica
    pos = Gira(-60) -- Representa signe '+' de la gramàtica
```

A continuació mostrem els diferents resultats:

```
13
Executant joc de proves 13 (fletxa 1)...
CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Gira 60.0 :#: CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: Gira 60.0 :#: CanviaCo
lor Color' 1.0 0.0 0.0 :#: Avança 10.0
```



Problema 14. branca

```
branca :: Int -> Comanda
```

A continuació es mostrarà la gramàtica per aquest problema, on a diferència del dos exercicis anteriors, sí que es farà servir la Comanda **Branca**. De fet, Podem veure que hi haurà branques dins de branques ja en el moment de fer la primera reescriptura.

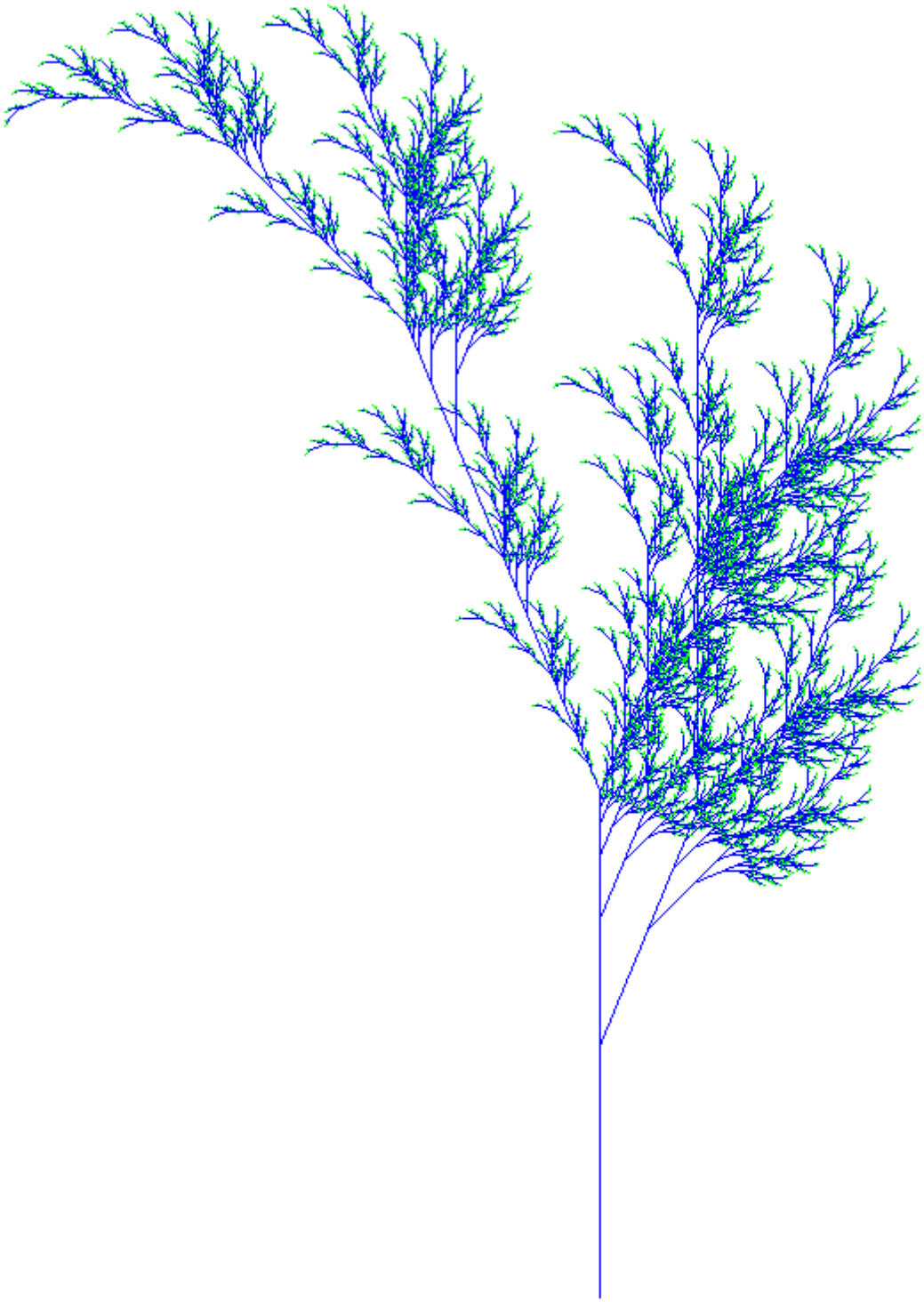
```
angle:          22.5
inici:          g
reescritura:    g → f-[[g]+g]+f[+fg]-g
                f → ff
```

El codi es mostrarà a continuació:

```
branca :: Int -> Comanda
branca lvl = g lvl
  where
    g 0 = CanviaColor verd :#: Avança 10 -- Si lvl = 0, retorna Avança
10
    g lvl = f (lvl-1) :#: neg :#: Branca (Branca (g (lvl-1)) :#: pos :#:
g (lvl-1)) :#: pos :#: f (lvl-1) :#: Branca (pos :#: f (lvl-1) :#: g
(lvl-1)) :#: neg :#: g (lvl-1) -- Va fent recursivitat amb les crides
fins que valgui 0
    f 0 = CanviaColor blau :#: Avança 10 -- Si lvl = 0, retorna Avança
10
    f lvl = f (lvl-1) :#: f (lvl-1) -- Va fent recursivitat amb les
crides fins que valgui 0
    neg = Gira (-22.5) -- Representa signe '-' de la gramàtica
    pos = Gira 22.5 -- Representa signe '+' de la gramàtica
```

A continuació mostrem els resultats d'execució i display:

```
14
Executant joc de proves 14 (branca 1)...
CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Gira -22.5 :#: Brancament Brancament CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#:
 Gira 22.5 :#: CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: Gira 22.5 :#: CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Brancam
ent Gira 22.5 :#: CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: Gira -22.5 :#: Can
viaColor Color' 0.0 0.0 1.0 :#: Avança 10.0
```



Millores

En aquest apartat realitzarem alguna de les propostes de millora de la pràctica. En concret, farem que el llapis dibuixi una figura que tingui branques.

Tot i que no ens ho demana, també crearem la nostra gramàtica per facilitar l'enteniment del que com evolucionaria la figura a cada iteració:

angle: 20º
inici: g
reescpritura: f → ff

g→ f[+g]f[-g]+g

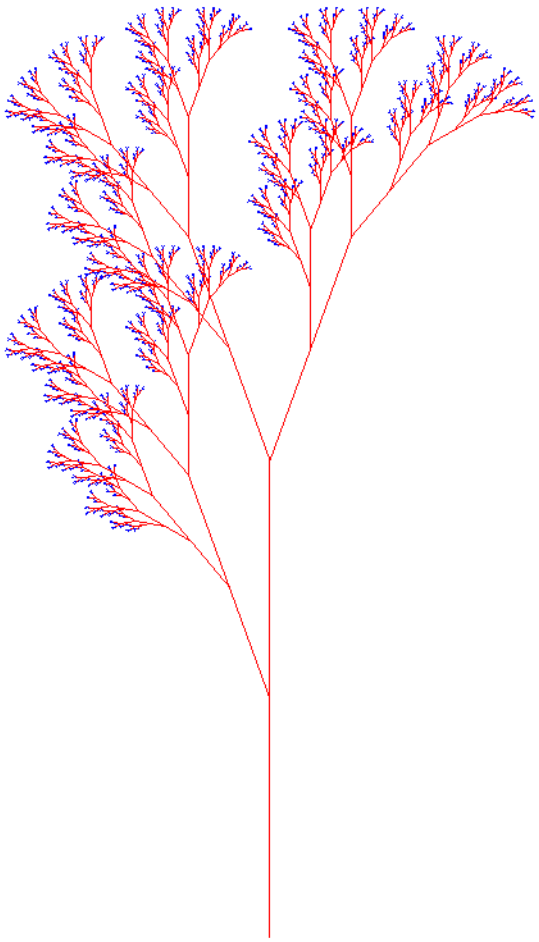
El codi de la funció seria aquest:

```
sticks :: Int -> Comanda
sticks lvl = g lvl
  where
    g 0 = CanviaColor blau :#: Avança 10 -- Si lvl = 0, retorna Avança 10
    g lvl = f (lvl-1) :#: Branca (pos :#: g (lvl-1)) :#: f (lvl-1) :#: Branca (neg :#: g (lvl-1)) :#: pos :#: g (lvl-1) -- Va fent recursivitat amb les crides fins que valgui 0
    f 0 = CanviaColor vermell :#: Avança 10 -- Si lvl = 0, retorna Avança 10
    f lvl = f (lvl-1) :#: f (lvl-1) -- Va fent recursivitat amb les crides fins que valgui 0
    neg = Gira (-20)  -- Representa signe '-' de la gramàtica
    pos = Gira 20     -- Representa signe '+' de la gramàtica
```

I el resultat de cridad la funció amb nivell 1 és la següent Comanda:

```
15
Executant joc de proves Opcional 1 (sticks 1)...
CanviaColor Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Brancament Gira 20.0 :#: CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: CanviaColor
 Color' 1.0 0.0 0.0 :#: Avança 10.0 :#: Brancament Gira -20.0 :#: CanviaColor Color' 0.0 0.0 1.0 :#: Avança 10.0 :#: Gira 20.0 :#: CanviaC
olor Color' 0.0 0.0 1.0 :#: Avança 10.0
```

I el dibuix resultant de fer-ho amb nivell 7 és aquest:



Bibliografia:

all:
http://zvon.org/other/haskell/Outputprelude/all_f.html
turtle:
https://hackage.haskell.org/package/turtle
wilberquito, session1:
https://github.com/wilberquito/Paradigms-and-Programming-Languages/blob/master/Haskell/Session1.short.md
wilberquito, session2:
https://github.com/wilberquito/Paradigms-and-Programming-Languages/blob/master/Haskell/Session2.md
wilberquito, session3:
https://github.com/wilberquito/Paradigms-and-Programming-Languages/blob/master/Haskell/Session3.md