

# MC202 — ESTRUTURAS DE DADOS

## Laboratório 05 — Fractais

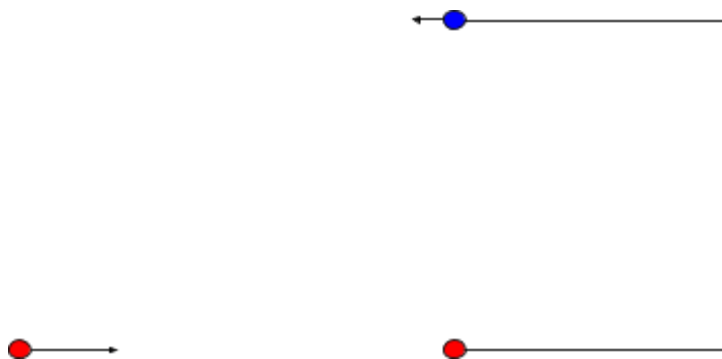
### Tarefa

Um fractal é um objeto geométrico caracterizado pela repetição de um padrão. Uma maneira de representar um tipo de padrão é usando um [L-Sistema](#), também chamado de Sistema de Lindenmayer. Esse sistema consiste em um alfabeto de símbolos e uma sequência de regras usadas para gerar cadeias de caracteres usando os símbolos do alfabeto, a partir de uma sequência inicial.

Turtle graphics é uma maneira de desenhar gráficos vetoriais a partir de uma localização e orientação, como se houvesse uma tartaruga com uma caneta em sua cabeça. A caneta registra o caminho percorrido pela tartaruga a partir de instruções que dizem qual direção seguir.

As instruções recebidas pela tartaruga podem ser representadas pelos símbolos usados nos L-Sistemas. Por exemplo, se o alfabeto for  $\{F, +, -\}$ , então F representa andar em linha reta por uma certa distância, + representa girar no sentido anti-horário por um certo ângulo e - representa girar no sentido horário por um certo ângulo.

Assim, considerando que a posição inicial da tartaruga está representada na imagem da esquerda como o ponto vermelho, que ela está direcionada para direita e que o ângulo de rotação escolhido é  $90^\circ$ , então após executar a sequência  $F+F+F$ , a tartaruga produzirá o desenho da direita, terminando no ponto azul e direcionada para a esquerda:

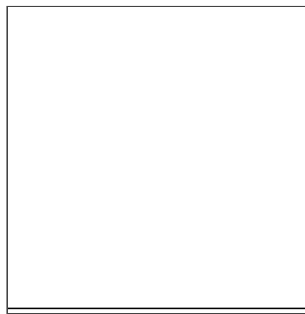


Para construir um fractal, há uma sequência inicial e um conjunto de regras que podem ser aplicadas recursivamente quantas vezes forem necessárias, substituindo-se os símbolos de acordo com a regra, como no exemplo abaixo:

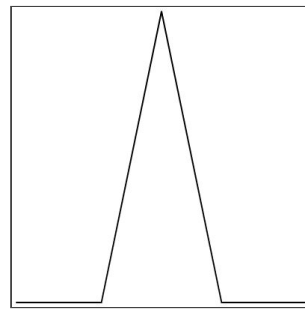
- **Alfabeto:**
  - **F** significa andar em uma linha reta
  - **+** significa virar 60° no sentido anti-horário
  - **-** significa virar 60° no sentido horário
- **Símbolo inicial:**
  - **F**
- **Regra:**
  - **F = F+F--F+F**

O processo de derivação acontece aplicando-se as regras:

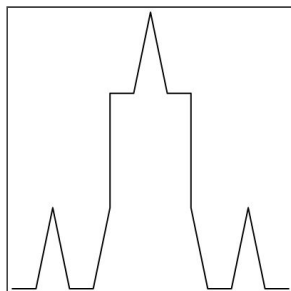
- Após 0 substituições:
  - F
- Após 1 substituição:
  - F+F--F+F
- Após 2 substituições:
  - F+F--F+F + F+F--F+F -- F+F--F+F + F+F--F+F
- Após 3 substituições:
  - F+F--F+F + F+F--F+F -- F+F--F+F + F+F--F+F +  
 F+F--F+F + F+F--F+F -- F+F--F+F + F+F--F+F --  
 F+F--F+F + F+F--F+F -- F+F--F+F + F+F--F+F +  
 F+F--F+F + F+F--F+F -- F+F--F+F + F+F--F+F



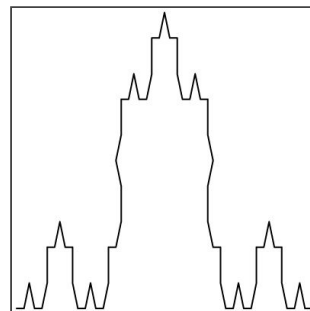
Inicial



1 substituição



2 substituições

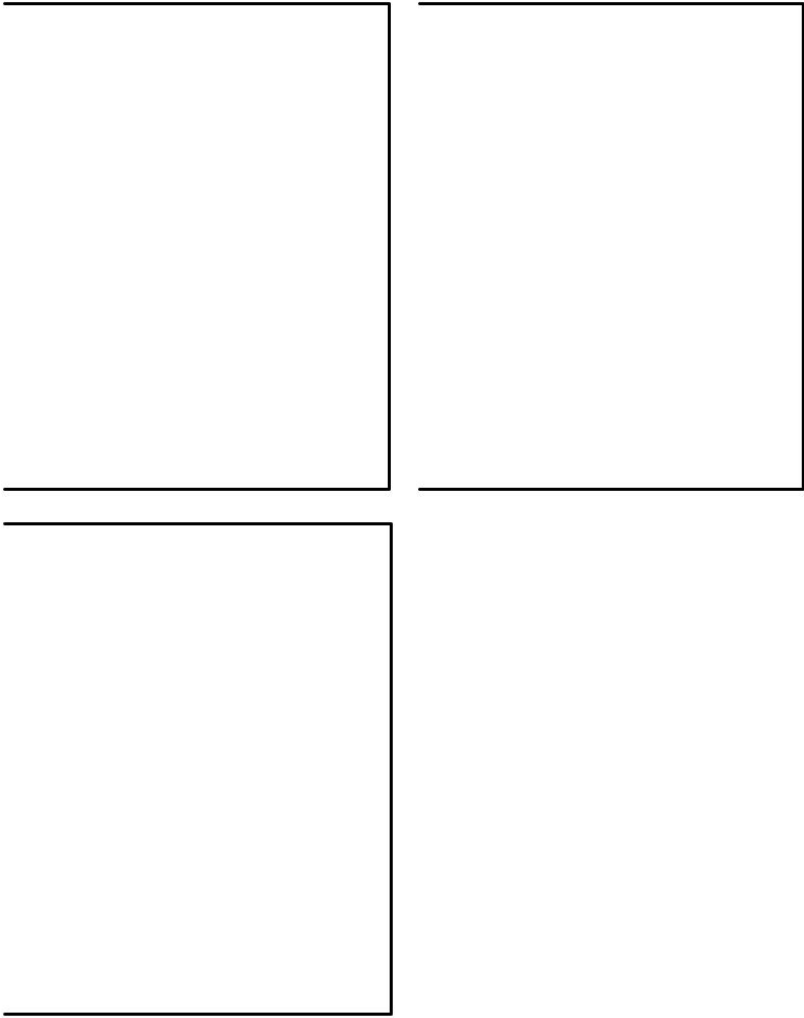


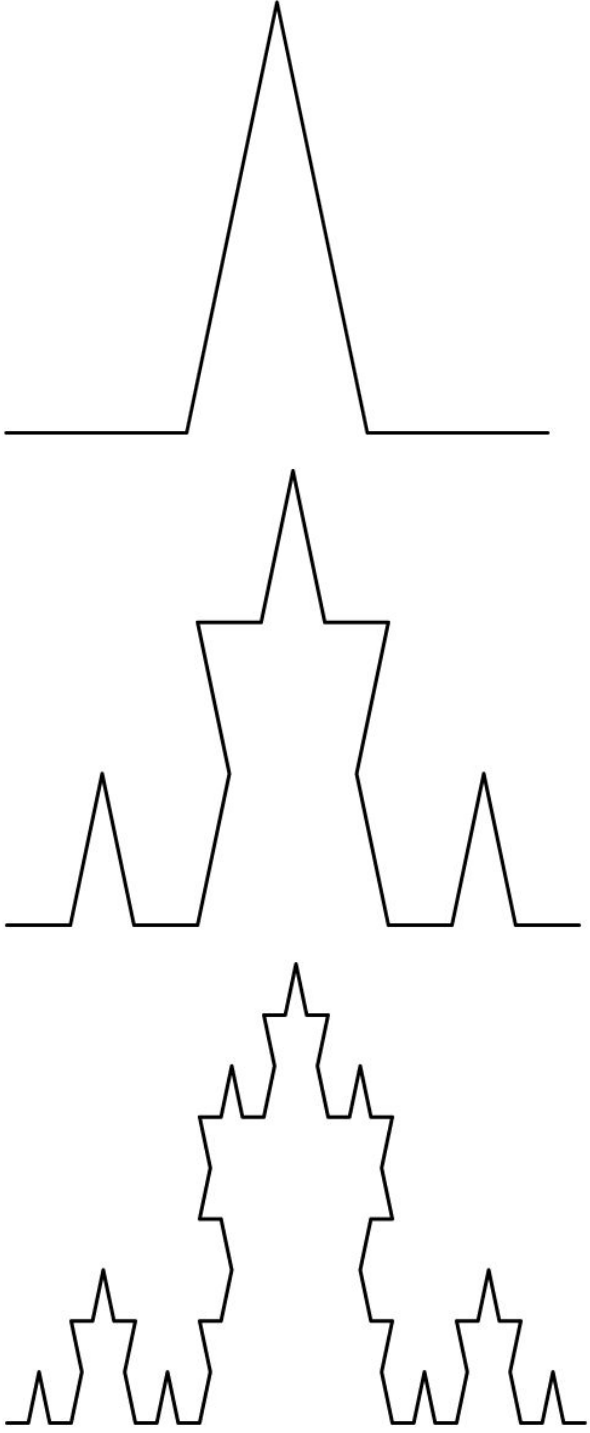
3 substituições

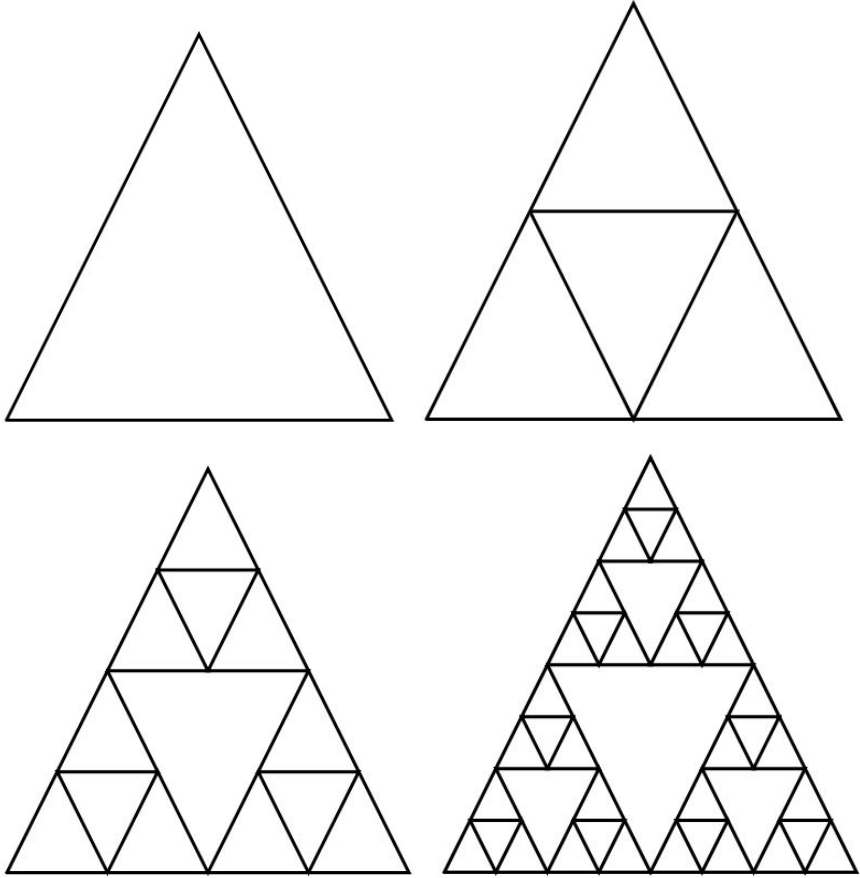
Sua tarefa será implementar funções recursivas que desenham diversos fractais diferentes. Para isso, você receberá uma pequena biblioteca de imagem que cria uma imagem seguindo a dinâmica do turtle graphics.

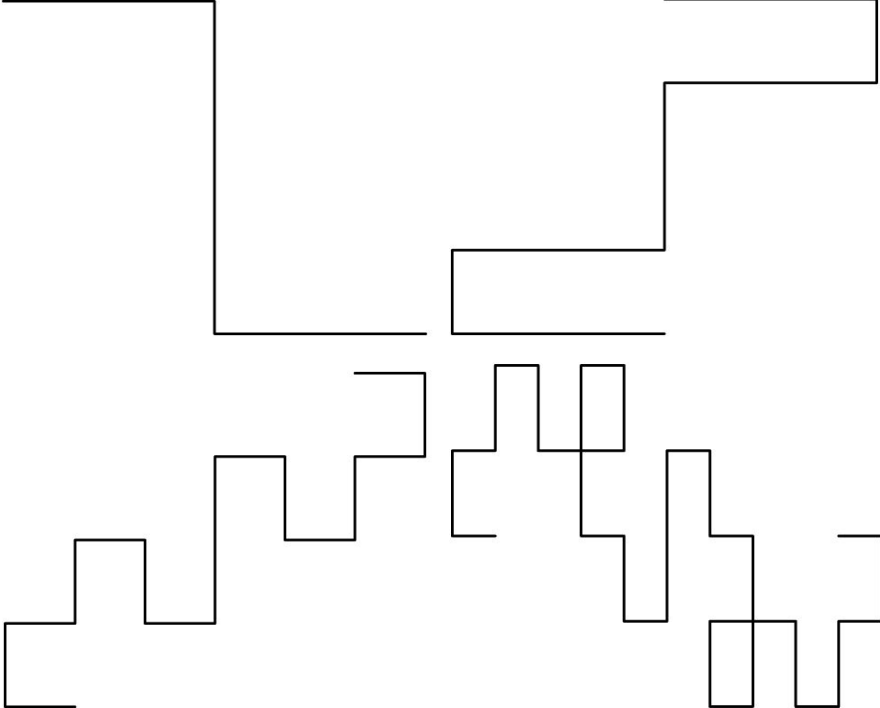
## Fractais

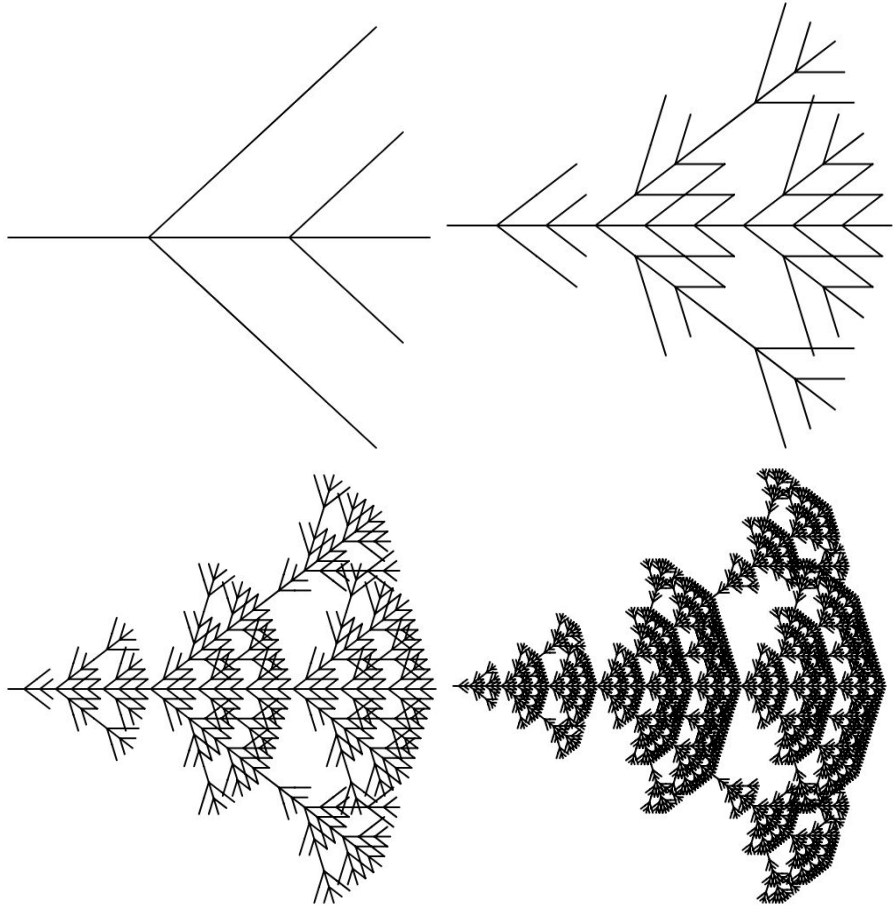
Os fractais que você deverá implementar são os seguintes. Observe que você deverá deduzir as regras de produção recursivas de alguns dos fractais a partir das imagens.

Nome:	• simples	
Sequência inicial:	• $F + F + F$	
Regras:	• $F = FF$	
Exemplo com ângulo $90^\circ$ :		

Nome:	<ul style="list-style-type: none"> <li>• koch</li> </ul>
Sequência inicial:	<ul style="list-style-type: none"> <li>• F</li> </ul>
Regras:	<ul style="list-style-type: none"> <li>• <math>F = F + F - F + F</math></li> </ul>
Exemplo com ângulo $60^\circ$ :	 <p>The image displays three stages of the Koch curve construction, illustrating the iterative process of adding detail to a fractal curve. The first stage is a simple horizontal line segment. The second stage shows the middle third of this segment being replaced by two sides of an equilateral triangle, creating a single triangular bump. The third stage shows this process repeated on each of the four segments of the second stage, resulting in a more complex, jagged shape with four distinct triangular bumps.</p>

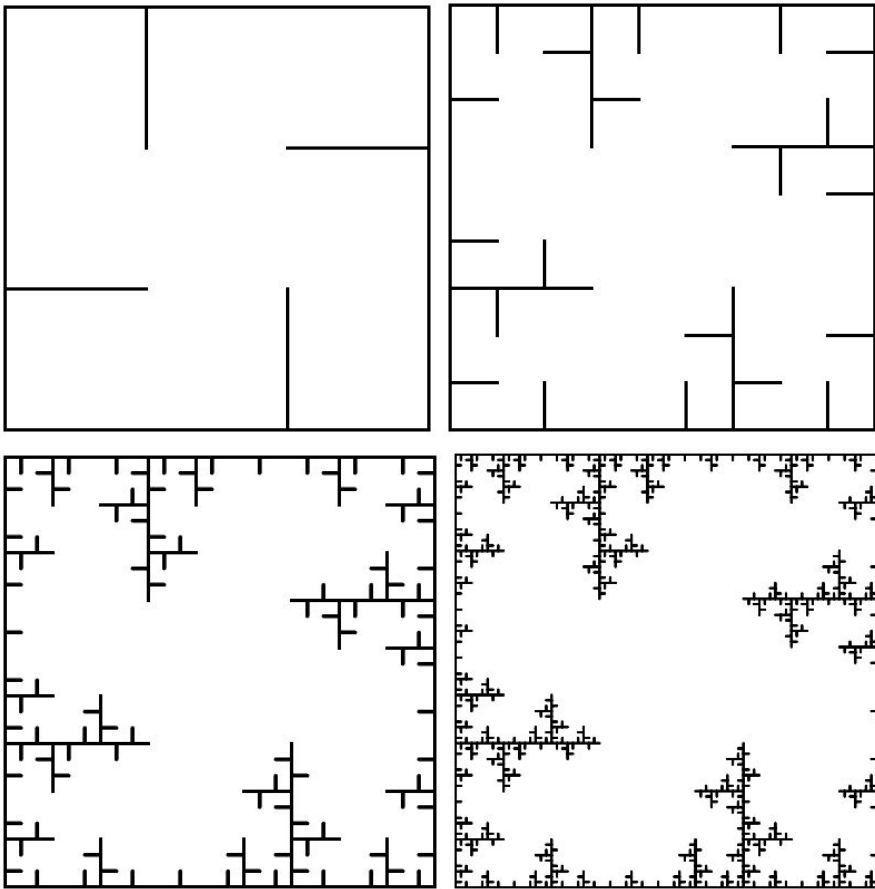
Nome:	<ul style="list-style-type: none"> <li>• sierpinski</li> </ul>
Sequência inicial:	<ul style="list-style-type: none"> <li>• F+G+G</li> </ul>
Regras:	<ul style="list-style-type: none"> <li>• <math>F = F+G-F-G+F</math></li> <li>• <math>G = GG</math></li> </ul>
Observações:	<ul style="list-style-type: none"> <li>• Os comandos da F e G significam andar em linha reta</li> </ul>
Exemplo com ângulo 120°:	 <p>The diagrams show the iterative construction of a Sierpinski triangle. The top-left diagram is a single large triangle. The top-right diagram shows the first iteration, where the central inverted triangle has been removed. The bottom-left diagram shows the second iteration, with more central inverted triangles removed. The bottom-right diagram shows the third iteration, resulting in a highly complex fractal structure with many small inverted triangles removed.</p>

Nome:	• dragao
Sequência inicial:	• FA
Regras:	<ul style="list-style-type: none"> <li>• <math>F = F-A</math></li> <li>• <math>A = F+A</math></li> </ul>
Observações:	• Os comandos da F e A significam andar em linha reta
Exemplo com ângulo 90°:	

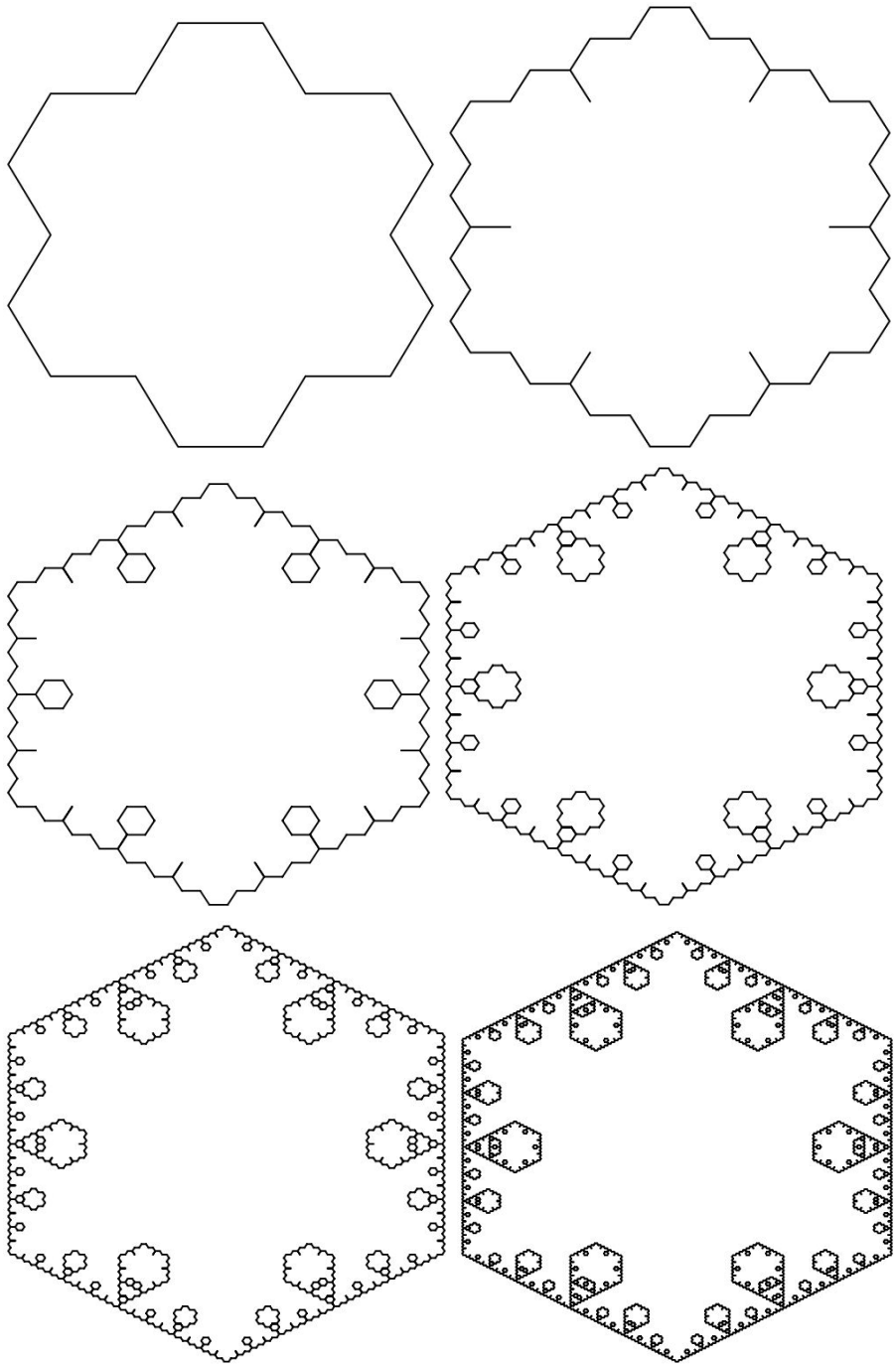
Nome:	<ul style="list-style-type: none"> <li>• arvore</li> </ul>
Sequência inicial:	<ul style="list-style-type: none"> <li>• F</li> </ul>
Regras:	<ul style="list-style-type: none"> <li>• <math>F = F[+FF][-FF]F[-F][+F]F</math></li> </ul>
Observações:	<ul style="list-style-type: none"> <li>• Os comandos entre [colchetes] significam que a tartaruga deve voltar para o ponto que estava quando entrou no colchete, depois de executar o movimento dentro dele</li> </ul>
Exemplo com ângulo 36°:	

## Implícitos

Encontrar a regra dos seguintes fractais apenas observando as imagens.

Nome:	<ul style="list-style-type: none"> <li>• gelo</li> </ul>
Sequência inicial:	<ul style="list-style-type: none"> <li>• <math>F+F+F+F</math></li> </ul>
Regras:	<ul style="list-style-type: none"> <li>• Encontrar a regra</li> </ul>
Observações:	<ul style="list-style-type: none"> <li>• Possui apenas os símbolos <math>\{F, +\}</math></li> <li>• Opcional, não conta nota</li> </ul>
Exemplo com ângulo $90^\circ$ :	



Nome:	<ul style="list-style-type: none"> <li>diamante</li> </ul>
Sequência inicial:	<ul style="list-style-type: none"> <li>F+F+F+F+F+F+</li> </ul>
Regras:	<ul style="list-style-type: none"> <li>Encontrar a regra</li> </ul>
Observações:	<ul style="list-style-type: none"> <li>Possui os símbolos {F, +, -}</li> <li>Há um único teste com esse fractal valendo nota</li> </ul>
Exemplo com ângulo 60°:	 <p>The image displays six stages of a fractal construction, arranged in a 3x2 grid. The fractal is a hexagon-like shape with a 60-degree angle. The stages show the iterative replacement of line segments with more complex patterns, resulting in a highly detailed, self-similar boundary. The fractal is composed of many small, interconnected hexagonal and triangular shapes, creating a complex, porous structure.</p>

## Entrada

A entrada contém 4 linhas:

- **nome do fractal** que deve ser desenhado
- **comprimento em pixels** que uma tartaruga deve andar ao receber o comando
- **ângulo** que a tartaruga deve girar ao receber o comando
- **número de substituições** realizadas no L-Sistema

## Saída

A saída será uma imagem formada por caracteres ascii no formato [pbm](#). As funções que geram a imagem serão fornecidas.

## Exemplo

### Entrada

```
simples
3
90
1
```

### Saída

```
P1
7 7
0000000
0111110
0000010
0000010
0000010
0111110
0000000
```

## Critérios específicos

Descreva os critérios específicos obrigatórios quando houver (qual estrutura usar, nomes dos arquivos, tempo, etc....)

- Você receberá os seguintes arquivos prontos e não deverá modificá-los:
  - imagem.h: interface da manipulação de imagem

- imagem.c: implementação das funções de manipulação de imagem
- Deverão ser submetidos os seguintes arquivos:
  - lab05.c: programa principal
- É obrigatório implementar cada solução usando **recursão**.
- Tempo máximo de execução: 20 segundos.
- Não há testes fechados para esse laboratório. As notas serão dadas com base nos testes 11 ao 20.

## Dicas

Exemplo de como usar a biblioteca imagem para criar a primeira iteração do fractal simples manualmente.

```
#include "imagem.h"

void main() {
    imagem_p img;

    img = criar_imagem();

    // simula os comandos F+F+F
    andar_cabeca(img, 3);
    girar_cabeca(img, 90);
    andar_cabeca(img, 3);
    girar_cabeca(img, 90);
    andar_cabeca(img, 3);

    imprimir_imagem(img);
    liberar_imagem(img);
}
```

Para salvar a posição da tartaruga e reutilizar depois:

```
#include "imagem.h"

void main() {
    imagem_p img;
    tartaruga cabeca;

    // Salva posição
    cabeca = obter_tartaruga(img);

    // Código
    ...

    // Retorna tartaruga para posição salva
    posicionar_tartaruga(img, cabeca);
}
```

## Testando

Neste laboratório, as saídas são imagens no formato .pbm. Na máquina virtual e em outros ambientes GNU/Linux, é provável ver a imagem apenas clicando-se no arquivo. No Windows, pode-se baixar gratuitamente um visualizador (experimente <https://www.irfanview.com/>).

Para compilar usando o Makefile fornecido e verificar se a solução está correta basta seguir o exemplo abaixo.

```
make
./lab05 < arq01.in > arq01.out.pbm
diff arq01.out.pbm arq01.res.pbm
```

onde arq01.in é a entrada (casos de testes disponíveis no SuSy) e arq01.out é a saída do seu programa. O Makefile também contém uma regra para testar todos os testes de uma vez; nesse caso, basta digitar:

```
make testar_tudo
```

Isso testará o seu programa com os casos abertos. Após o prazo, os casos de teste fechados serão liberados e podem ser baixados digitando-se:

```
make baixar_fechados
```

Para ver quanto tempo seu programa demora em cada teste, digite:

```
make tempo
```

# Observações gerais

No SuSy, haverá 3 tipos de tarefas com siglas diferentes para cada laboratório de programação. Todas possuirão os mesmos casos de teste. As siglas são:

1. **DRAFT:** Esta tarefa serve para testar o programa no SuSy antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém arquivos submetidos aqui não serão corrigidos.
2. **ENTREGA:** Esta tarefa tem limite de *uma única submissão* e serve para entregar a versão final dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa: submeta aqui quando não for mais fazer alterações no seu programa.
3. **FORAPRAZO:** Esta tarefa tem limite de *uma única submissão* e serve para entregar a versão final após o prazo estabelecido para o laboratório, mas com nota reduzida (conforme a ementa). O envio nesta tarefa irá substituir a nota obtida na tarefa ENTREGA apenas se o aluno tiver realizado as correções sugeridas no feedback ou caso não tenha enviado anteriormente em ENTREGA.

Observações sobre SuSy:

- Versão do GCC: gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)
- Flags de compilação:  
`-std=c99 -Wall -Werror -Werror=vla -pedantic-errors -g -lm`

Além das observações acima, esse laboratório será avaliado pelos critérios gerais:

- Indentação de código e outras boas práticas, tais como:
  - uso de comentários (úteis e apenas quando forem relevantes);
  - código simples e fácil de entender;
  - sem duplicidade (partes que fazem a mesma coisa).
- Organização do código:
  - tipos de dados criados pelo usuário e funções bem definidas e tão independentes quanto possível.
- Corretude do programa:
  - programa correto e implementado conforme solicitado no enunciado;
  - inicialização de variáveis sempre que for necessário;
  - dentre outros critérios.