Descrição de cada uma das três implementações e das estratégias escolhidas para tal (recursividade, iteratividade):

Os três fractais foram implementados de maneira semelhante, com apenas algumas pequenas diferenças sutis entre eles.

A construção dos programas foi feita de forma recursiva, pois essa abordagem simplifica o código e evita a criação de estágios intermediários durante a obtenção do resultado final e, consequentemente, trata-se de uma solução geralmente mais rápida e eficiente do que o método iterativo.

Contudo, ela apresenta maior consumo de memória e está sujeita a um limite de profundidade recursiva. Caso o estágio do fractal solicitado pelo usuário seja muito grande, pode haver um erro de stack overflow.

Floco de Neve Onda Senoidal 1 de Koch

A implementação da Onda Senoidal 1 de Koch foi feita de maneira recursiva e aceita três parâmetros:

- A ordem do fractal, que define a profundidade que ainda falta percorrer da recursão;
- O índice, que guarda em qual caractere da sequência a função atual está;
- Um ponteiro para um arquivo onde a saída será escrita.

A implementação é dividida em dois casos principais:

1. **Caso base**: Quando a ordem é igual a 1, o fractal atinge o nível mais simples, e a sequência base **koch** é diretamente escrita no arquivo por meio do comando '**fprintf'**, sem precisar passar por mais um passo recursivo. Assim, esse caso encerra a recursão.
2. **Caso recursivo**: A função percorre cada caractere da string `koch`. Se o caractere for F, a função chama a si mesma, reduzindo a ordem do fractal. Para os outros caracteres, como `-`, `+`, e espaços, esses são diretamente gravados no arquivo, uma vez que não possuem uma regra de recursão.

Preenchimento de Espaço de Hilbert

A implementação do Preenchimento de Espaço de Hilbert foi feita de maneira recursiva e recebe quatro parâmetros:

- **A ordem do fractal**, que indica a profundidade restante da recursão;
- **O índice**, que representa a posição atual na sequência que está sendo percorrida;
- **O pai**, que determina se o fractal segue a regra de substituição para `X` ou para `Y`;

- **Um ponteiro para um arquivo**, onde a saída será gravada.

A implementação é estruturada em dois casos principais:

1. **Caso base**: Quando a ordem do fractal é 1, a função escreve a sequência base já diretamente no arquivo - a sequência base (hilbert[X] ou hilbert[Y]) é definida por meio do valor da variável pai - e encerra a recursão .
    - Para o símbolo X, a função percorre a sequência associada no array hilbert[X] . Os caracteres X e Y não são impressos.
    - Para o símbolo Y, a função percorre a sequência associada no array hilbert[Y] . Os caracteres X e Y não são impressos.

2. **Caso recursivo**: Enquanto percorre a sequência associada ao pai (X ou Y), a função avalia cada caractere:
    - Se o caractere for X ou Y, a função chama a si mesma recursivamente, reduzindo a ordem e ajustando o pai para o símbolo correspondente.
    - Para os demais caracteres, como −, +, ou F, estes são diretamente escritos no arquivo sem modificações.

Ponta de Flecha de Sierpinski

A implementação da Ponta de Flecha de Sierpinski foi feita de maneira muito semelhante a do Preenchimento do Espaço de Hilbert, com a única diferença de que, após a sua execução, um caractere "F" a mais é acrescentado. Isso decorre do axioma gerador do fractal, no caso, YF.

A função geradora recebe quatro parâmetros:

- **A ordem do fractal**, que indica a profundidade restante da recursão;
- **O índice**, que representa a posição atual na sequência que está sendo percorrida;
- **O pai**, que determina se o fractal segue a regra de substituição para X ou para Y;
- **Um ponteiro para um arquivo**, onde a saída será gravada.

A implementação é estruturada em dois casos principais:

- **Caso base**: Quando a ordem do fractal é 1, a função escreve a sequência base já diretamente no arquivo - a sequência base (sierpinski[X] ou sierpinski[Y]) é definida por meio do valor da variável pai - e encerra a recursão .
    - Para o símbolo X, a função percorre a sequência associada no array sierpinski[X] . Os caracteres X e Y não são impressos.

    - Para o símbolo Y, a função percorre a sequência associada no array sierpinski[Y] . Os caracteres X e Y não são impressos.

- **Caso recursivo**: Enquanto percorre a sequência associada ao pai (X ou Y), a função avalia cada caractere:
    - Se o caractere for X ou Y, a função chama a si mesma recursivamente, reduzindo a ordem e ajustando o pai para o símbolo correspondente.
    - Para os demais caracteres, como −, +, ou F, estes são diretamente escritos no arquivo sem modificações.

Apresentação da complexidade de seus algoritmos considerando a notação assintótica mais precisa possível;

Floco de Neve Onda Senoidal 1 de Koch: Θ (6^n)

Preenchimento de Espaço de Hilbert: Θ (4^n)

Ponta de Flecha de Sierpinski: Θ (3^n)

Para cada um dos fractais implementados, apresente a equação de recorrência para calcular a quantidade de segmentos F gerados e a quantidade de símbolos existentes em cada estágio;

Floco de Neve Onda Senoidal 1 de Koch

Símbolos:

$t(n) = 6 * t(n-1) + 6$

$t(0) = 1$

Segmentos F:

$t(n) = 6 * t(n-1)$

$t(0) = 1$

Preenchimento de Espaço de Hilbert

Símbolos:

$t(n) = 4 * t(n-1) + 7$

$t(0) = 1$

Segmentos F:

$t(n) = 4 * t(n-1) + 3$

$t(0) = 0$

Ponta de Flecha de Sierpinski

Símbolos:

$t(n) = 3 * t(n-1) + 2$

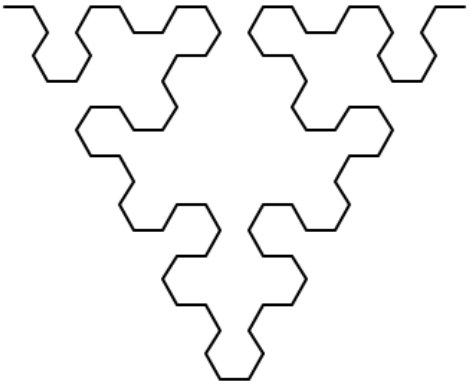
$t(0) = 2$

Segmentos F:

$t(n) = 3 * t(n-1)$

$t(0) = 1$

Figuras dos primeiros quatro estágios do fractal proposto por você (fractal iii). Além disso, é necessário incluir uma descrição detalhada do processo de criação e do software utilizado para gerar as imagens.



Para gerar as imagens, foi utilizado um script desenvolvido em Python, responsável por ler o conteúdo do arquivo .txt, no caso, iii.txt, e, com base nele, utilizar a biblioteca gráfica Turtle para desenhar os fractais na tela.

1. **Leitura dos Dados do Arquivo de Entrada (iii.txt)**
   O arquivo iii.txt contém a sequência de comandos necessários para a construção do fractal (F+-FXY …). O código abre o arquivo, processa os dados e gera uma imagem correspondente.

2. **Uso da Biblioteca Turtle**
   A biblioteca Turtle, integrada ao Python, foi escolhida por sua simplicidade e rápida curva de aprendizado.

3. **Captura das Imagens**
   Após a conclusão de cada estágio, um print screen foi utilizado para capturar as imagens diretamente da tela.

Ferramentas e Software Utilizados

1. **Python (versão 3.x)**
   - Linguagem de programação utilizada para escrever o script.
2. **Biblioteca Turtle**
   - Ferramenta gráfica usada para desenhar as linhas do fractal.
3. **Print Screen**
   - As capturas foram recortadas e organizadas para melhor apresentação.