

# AULA 5

# Hoje

- ▶ mais **recursão**
- ▶ argumentos na **linha de comando**
- ▶ **registros e estruturas**

# Curvas de Hilbert

Niklaus Wirth  
*Algorithms and Data Structures*  
Prentice Hall, 1986.

# Curvas de Hilbert

As curvas a seguir seguem um certo **padrão regular** e podem ser desenhadas na tela sobre o controle de um programa.

O objetivo é descobrir o **esquema de recursão** para construir tais curvas.

Estes padrões serão chamados de  $H_0, H_1, H_2 \dots$

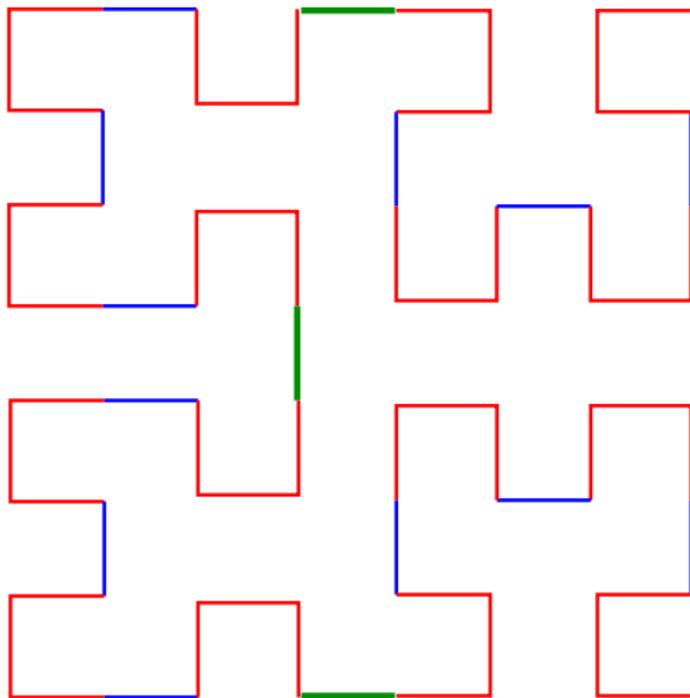
Cada  $H_i$  denomina a **curva de Hilbert** de **ordem  $i$** , em homenagem a seu inventor, o matemático *David Hilbert*.

$H_1$





$H_3$





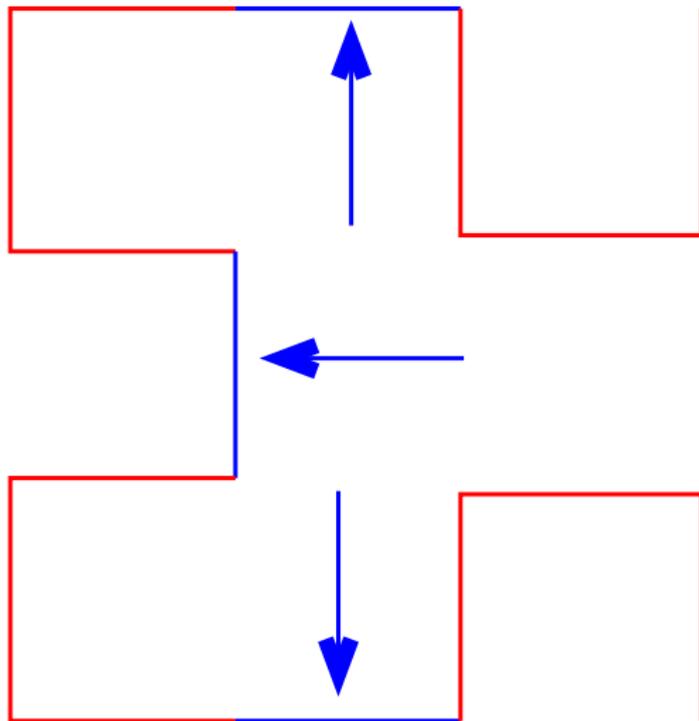
# Padrão

As figuras mostram que  $H_{i+1}$  é obtida pela composição de 4 instâncias de  $H_i$  de metade do tamanho e com a rotação apropriada, ligadas entre si por meio de 3 linhas de conexão.

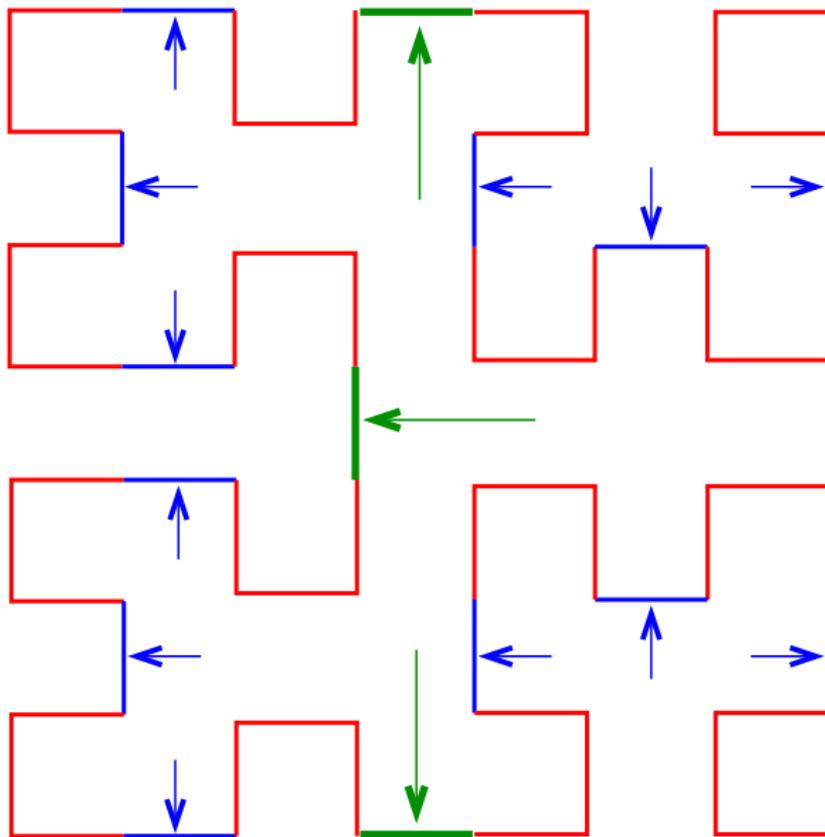
Por exemplo:

- ▶  $H_1$  é formada por 4  $H_0$  (vazio) conectados por 3 linhas.
- ▶  $H_2$  é formada por 4  $H_1$  conectados por 3 linhas
- ▶  $H_3$  é formada por 4  $H_2$  conectados por 3 linhas

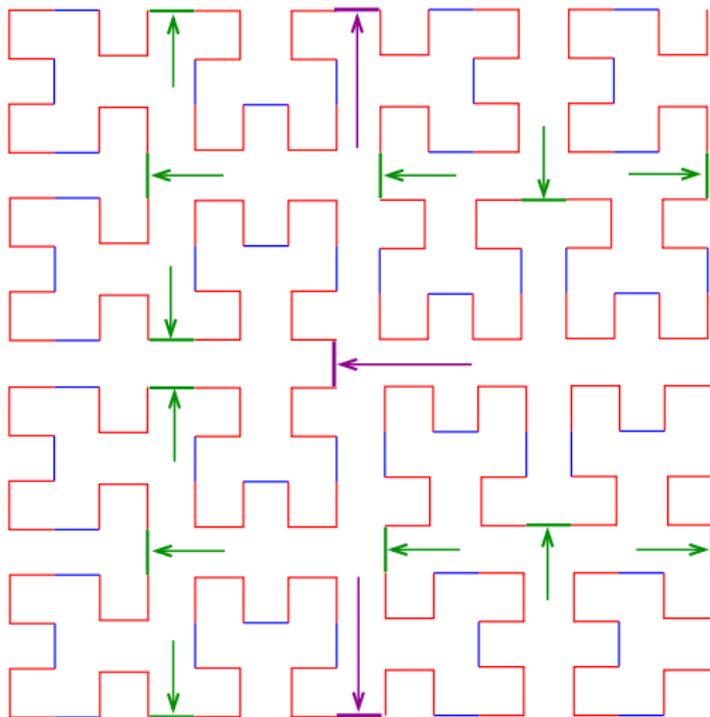
H<sub>2</sub>



$H_3$



$H_4$



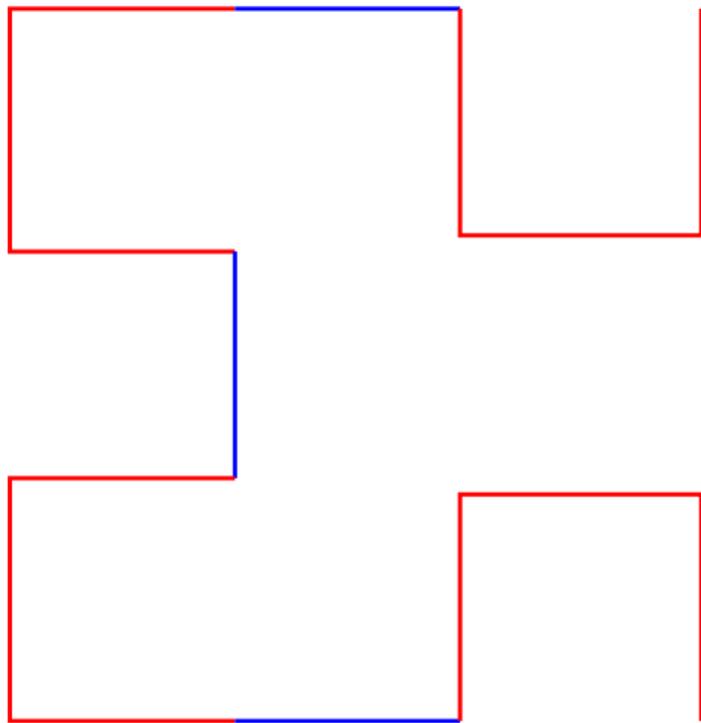
## Partes da curva

Para ilustrar, denotaremos as quatro possíveis instâncias por **A**, **B**, **C** e **D**:

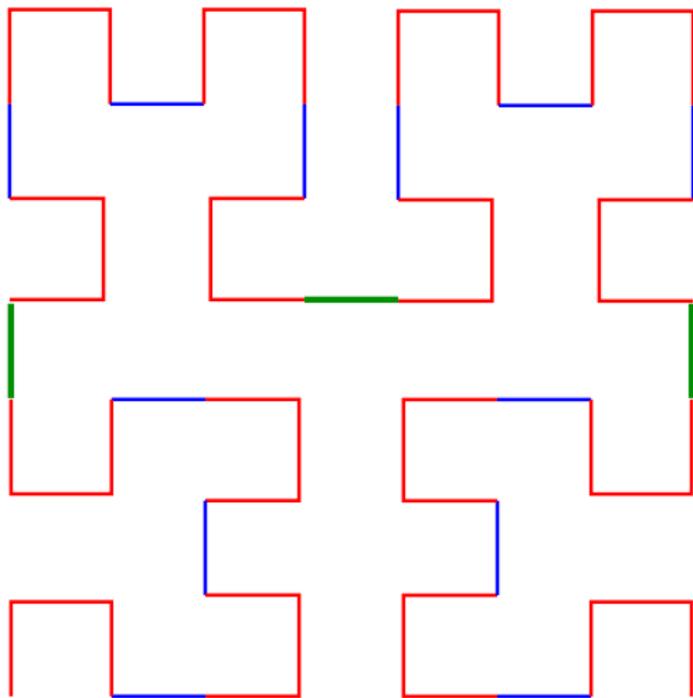
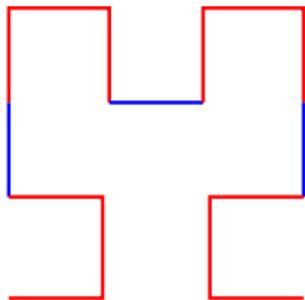
- ▶ **A** será o padrão que tem a “abertura” para **direita**;
- ▶ **B** será o padrão que tem a “abertura” para **baixo**;
- ▶ **C** será o padrão que tem a “abertura” para **esquerda**; e
- ▶ **D** será o padrão que tem a “abertura” para **cima**.

Representaremos a chamada da função que desenha as interconexões por meio das setas  $\uparrow$ ,  $\downarrow$ ,  $\leftarrow$ ,  $\rightarrow$ .

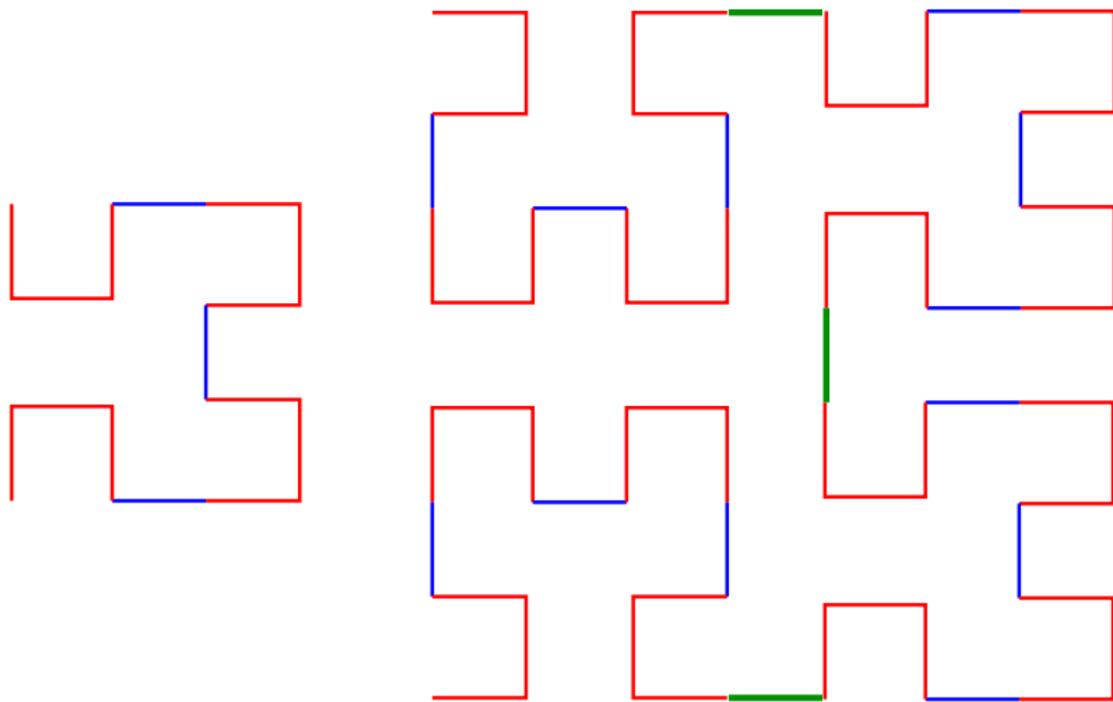
$A_1$  e  $A_2$



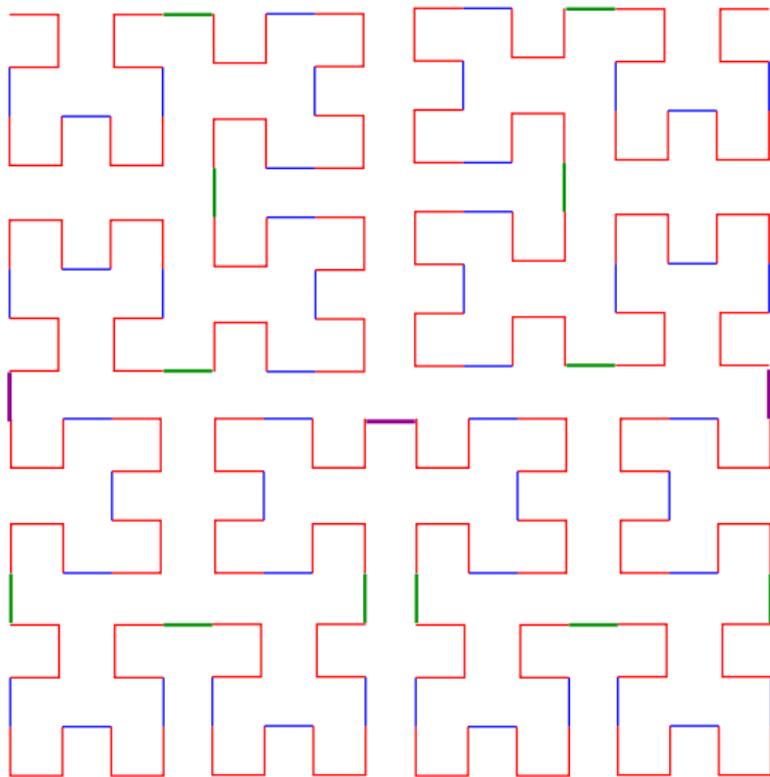
$B_2$  e  $B_3$



$C_2$  e  $C_3$



$D_4$



## Esquema recursivo

Assim, surge o seguinte esquema recursivo:

$$\begin{array}{l} A_k : D_{k-1} \leftarrow A_{k-1} \downarrow A_{k-1} \rightarrow B_{k-1} \\ B_k : C_{k-1} \uparrow B_{k-1} \rightarrow B_{k-1} \downarrow C_{k-1} \\ C_k : B_{k-1} \rightarrow C_{k-1} \uparrow C_{k-1} \leftarrow D_{k-1} \\ D_k : A_{k-1} \downarrow D_{k-1} \leftarrow D_{k-1} \uparrow C_{k-1} \end{array}$$

Para desenhar os segmentos utilizaremos a chamada de uma função

`linha(x,y,direcao,comprimento)`

que “**move um pincel**” da posição  $(x,y)$  em uma dada **direcao** por um certo **comprimento**.

```
typedef enum {DIREITA, ESQUERDA, CIMA, BAIXO} Direcao;
void linha(int *x, int *y, Direcao direcao,
           int comprimento) {
    switch (direcao) {
        case DIREITA : *x = *x + comprimento;
                       break;
        case ESQUERDA : *x = *x - comprimento;
                       break;
        case CIMA : *y = *y + comprimento;
                   break;
        case BAIXO : *y = *y - comprimento;
                   break;
    }
    desenhaLinha(*x, *y);
}
```

$A_k$

```
void
a(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        d(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        a(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        a(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        b(k-1, x, y, comprimento);
    }
}
```

$B_k$ 

```
void
b(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        c(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        b(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        b(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        a(k-1, x, y, comprimento);
    }
}
```

$C_k$

```
void
c(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        b(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        c(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        c(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        d(k-1, x, y, comprimento);
    }
}
```

$D_k$

```
void
d(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        a(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        d(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        d(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        c(k-1, x, y, comprimento);
    }
}
```

# Argumentos na linha de comando

# Argumentos na linha de comando

Quando `main` é chamada, ela recebe dois argumentos:

- ▶ `argc` ('c' de *count*) é o número de argumentos que o programa recebeu na linha de comando; e
- ▶ `argv[ ]` é um vetor de *strings* contendo cada um dos argumentos.

Por convenção `argv[0]` é o nome do programa que foi chamado. Assim, `argc` é sempre pelo menos 1.

# Argumentos na linha de comando

Por exemplo, na chamada

```
meu_prompt> echo Hello World!
```

- ▶ `argc = 3`
- ▶ `argv[0] = "echo"`
- ▶ `argv[1] = "Hello"`
- ▶ `argv[2] = "World!"`

# Argumentos na linha de comando

Na chamada

```
meu_prompt> gcc echo.c -o echo
```

- ▶ `argc` = 4
- ▶ `argv[0]` = "gcc"
- ▶ `argv[1]` = "echo.c"
- ▶ `argv[2]` = "-o"
- ▶ `argv[3]` = "echo"

## echo.c

```
#include <stdio.h>
int
main(int argc, char *argv[])
{
    int i;
    for (i = 1; i < argc; i++)
        printf("%s ", argv[i]);
    printf("\n");
    return 0;
}
```

## echo.java

```
public class echo {  
    public static void main(String argv[])  
    {  
        for (int i=0; i < argv.length; i++)  
            System.out.print(argv[i] + );  
        System.out.print("\n");  
        System.exit(0);  
    }  
}
```