AULA 5

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.

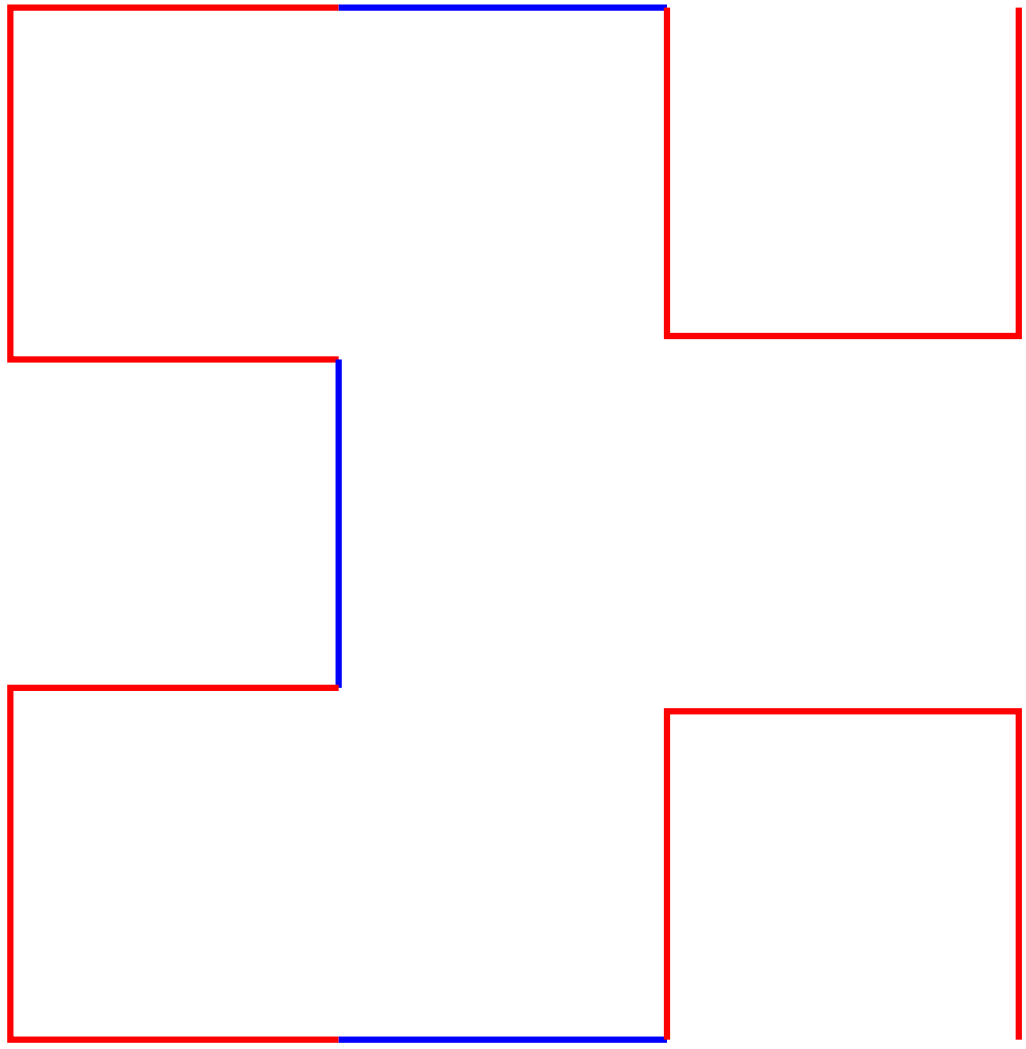
O objetivo é descobrir o esquema de recursão para construir tais curvas.

Estes padrões serão chamados de $\mathtt{H}_0, \mathtt{H}_1, \mathtt{H}_2 \ldots$.

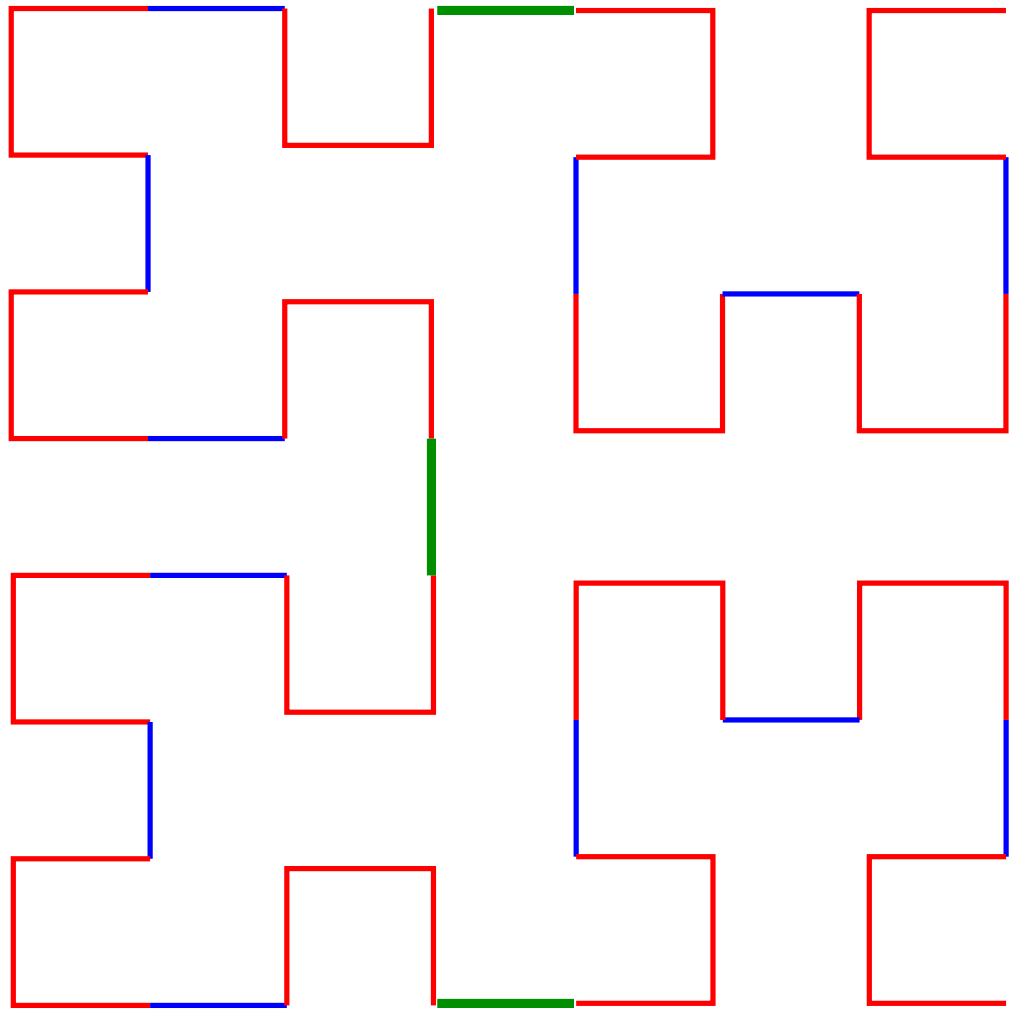
Cada $\mathtt{H}_\mathtt{i}$ denomina a **curva de Hilbert** de ordem $\mathtt{i}$, em homenagem a seu inventor, o matemático *David Hilbert*.

H_1

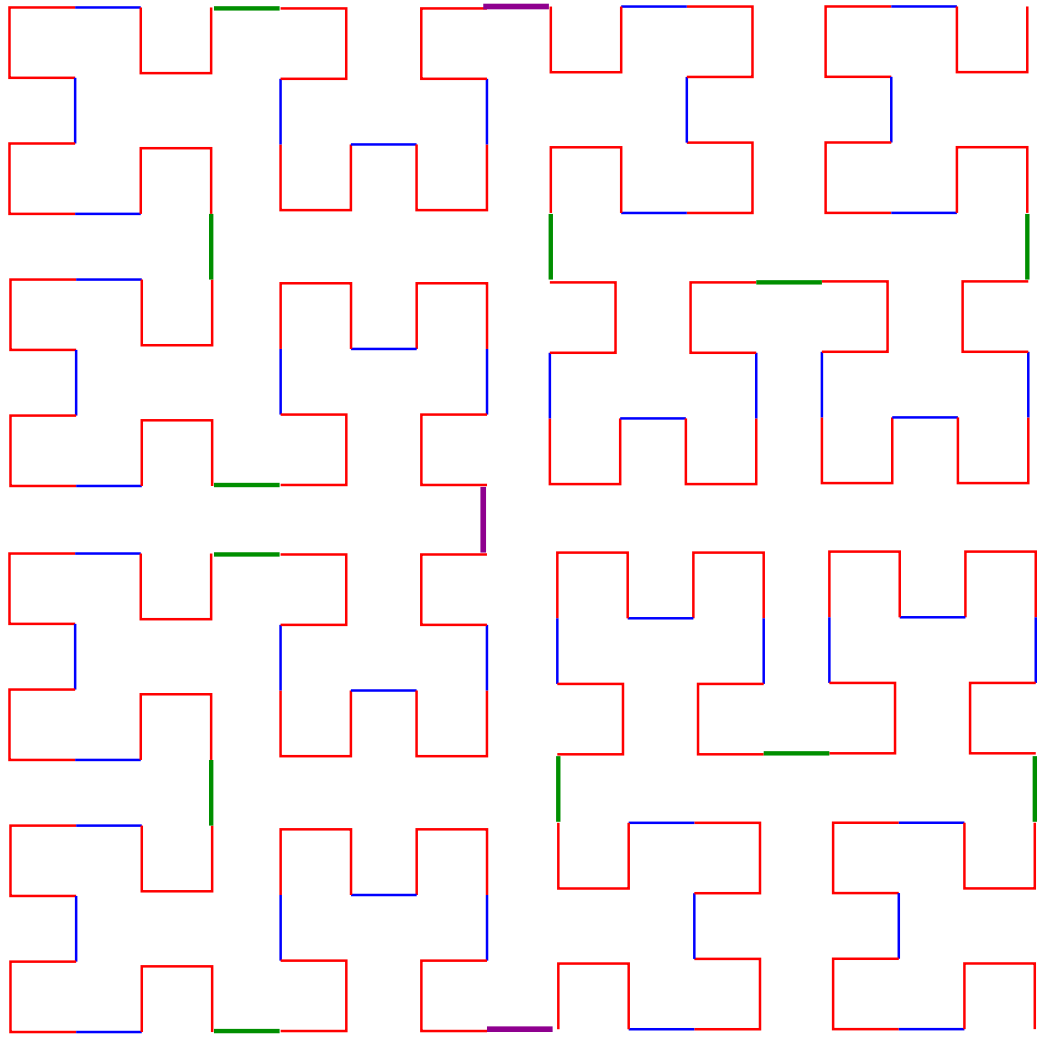
H_2



H_3



H_4



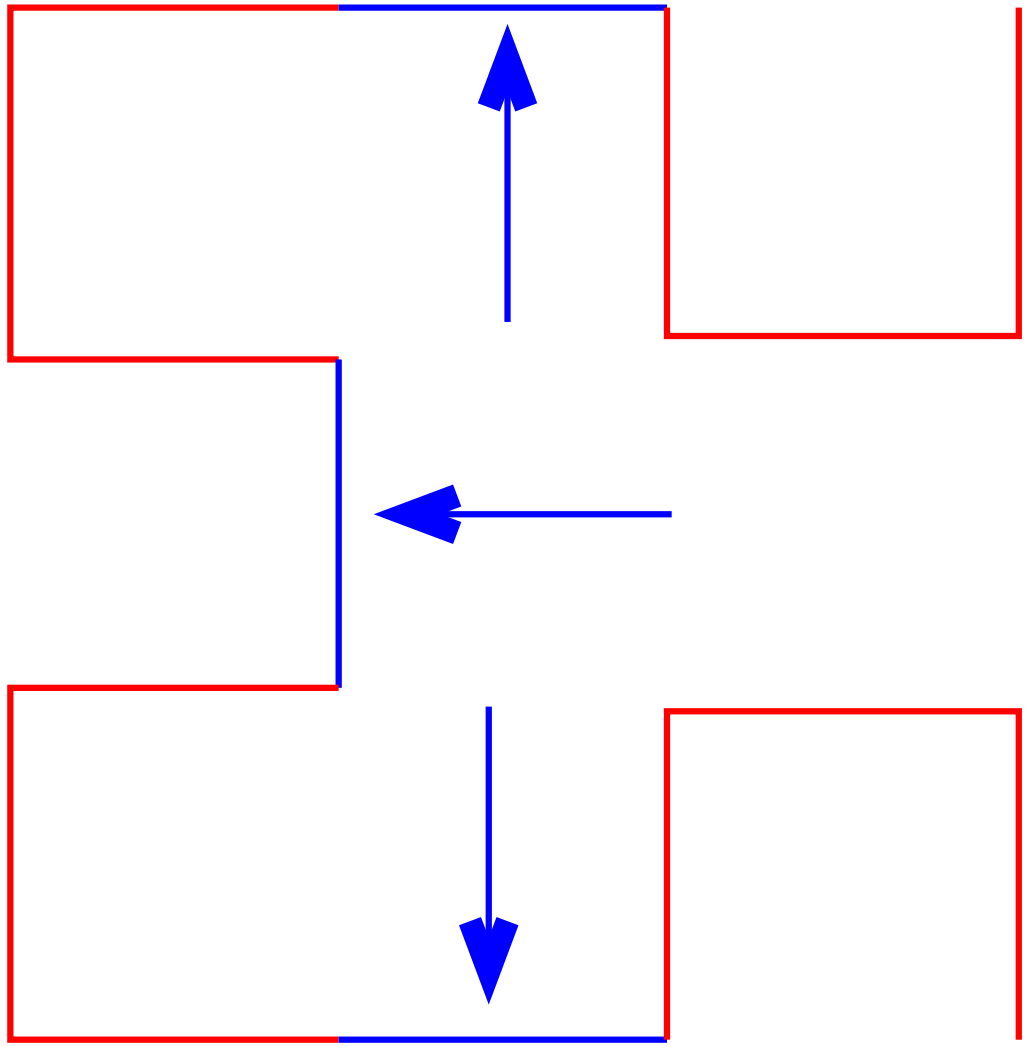
Padrão

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.

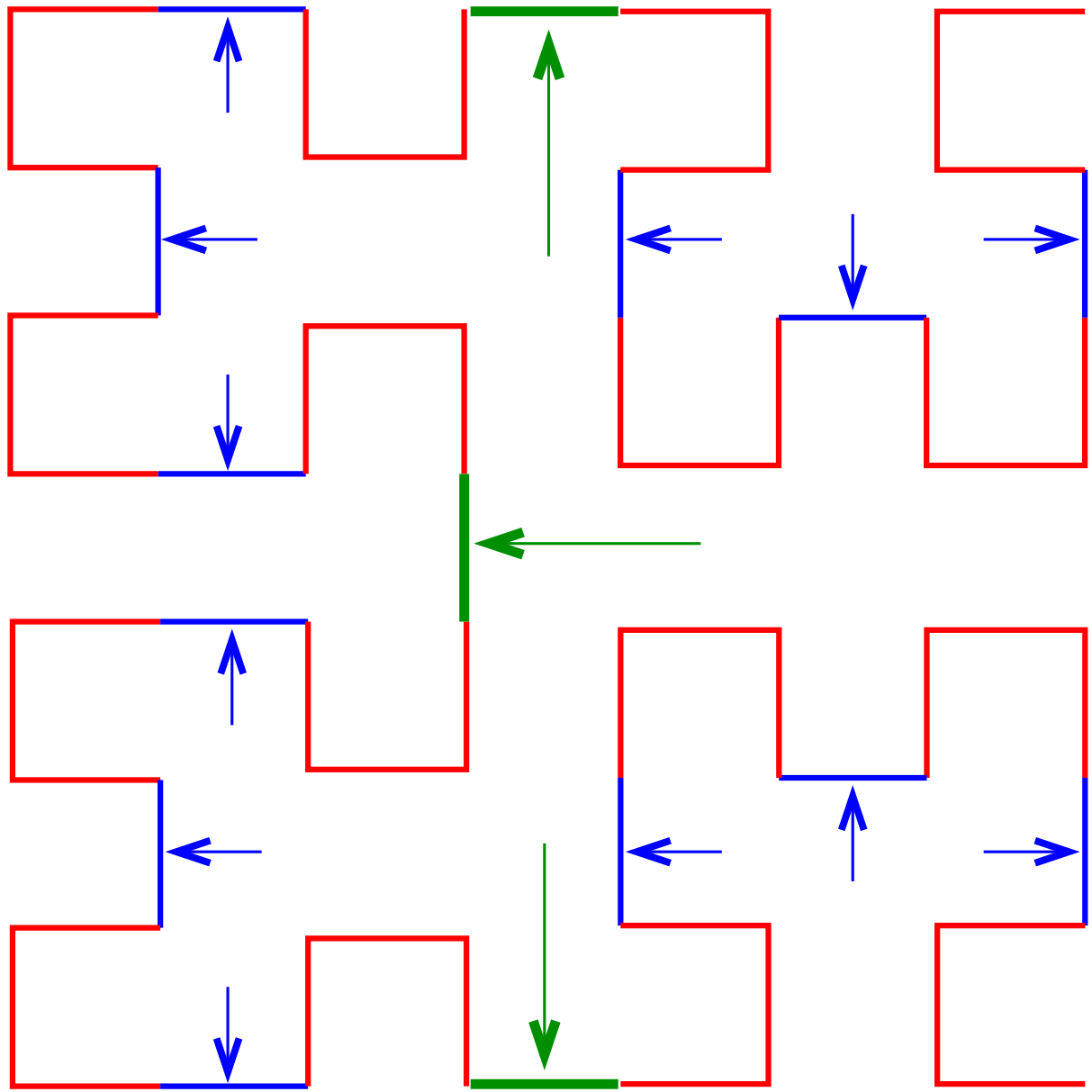
Por exemplo:

- 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

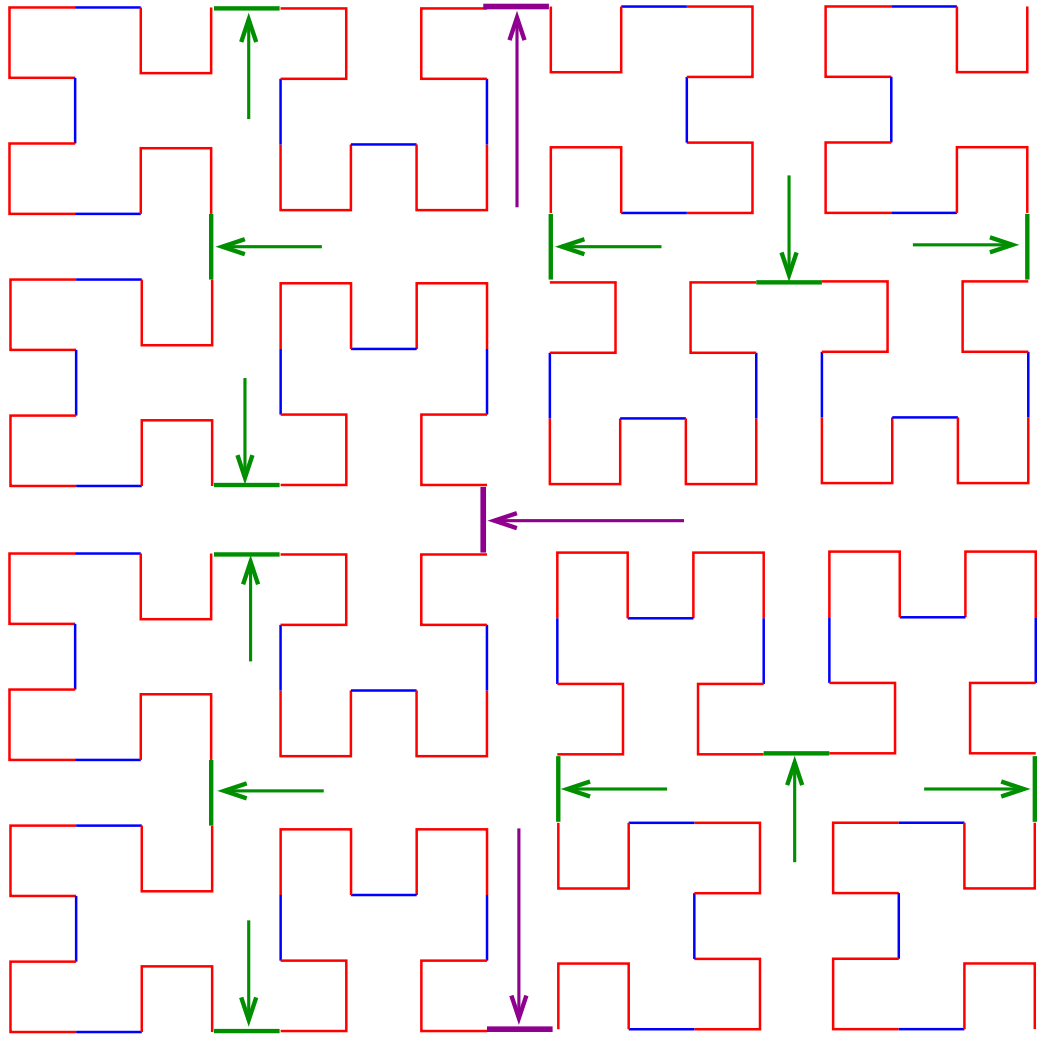
H_2



H_3



H_4



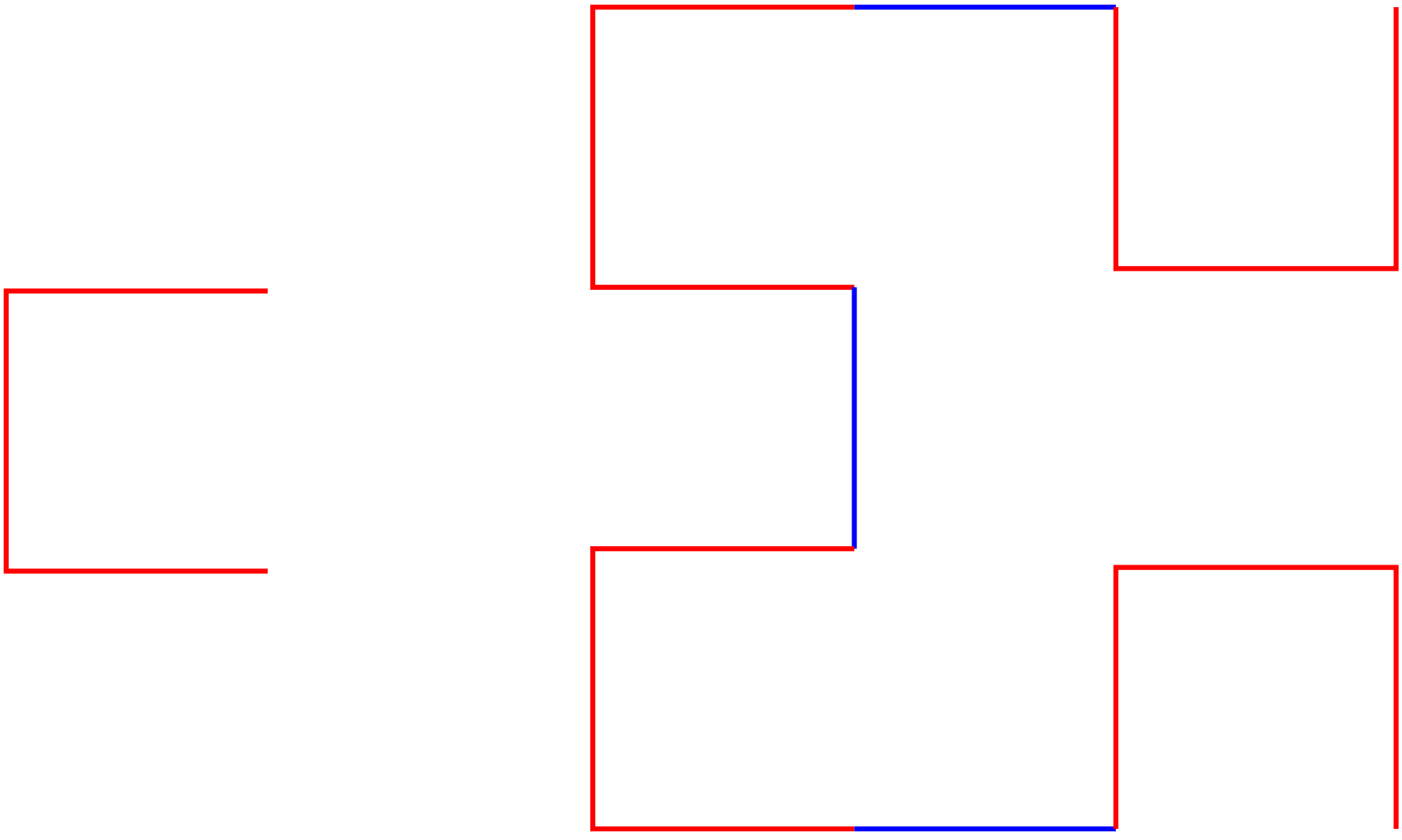
Partes da curva

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

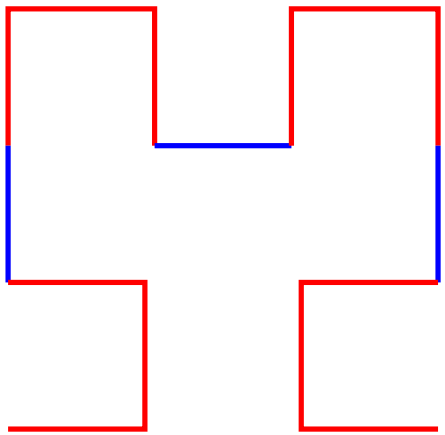
- A será o padrão que tem a "abertura" para direita;
- B será o padrão que tem a "abertura" para baixo;
- C será o padrão que tem a "abertura" para esquerda; e
- D será o padrão que tem a "abertura" para cima.

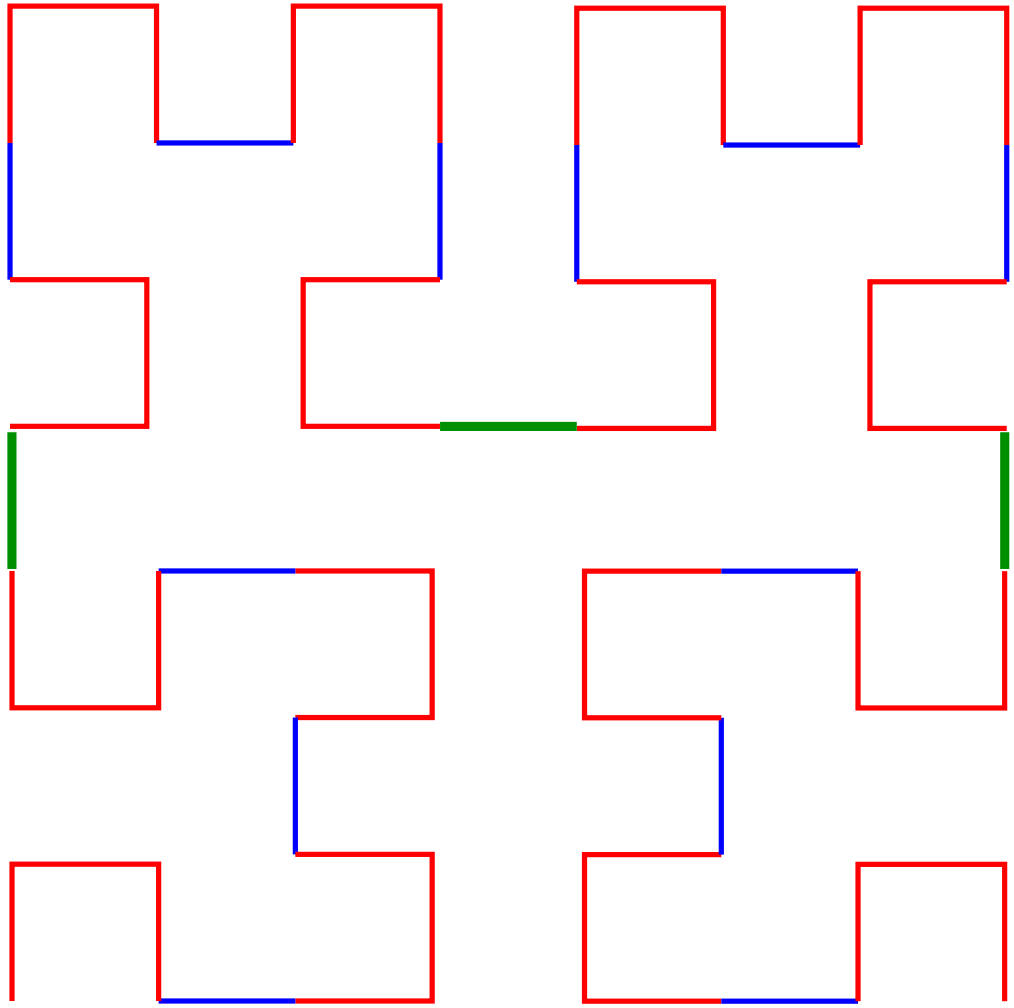
Representaremos a chamada da função que desenha as interconexões por meio da setas $\uparrow, \downarrow, \leftarrow, \rightarrow$.

A_1 e A_2

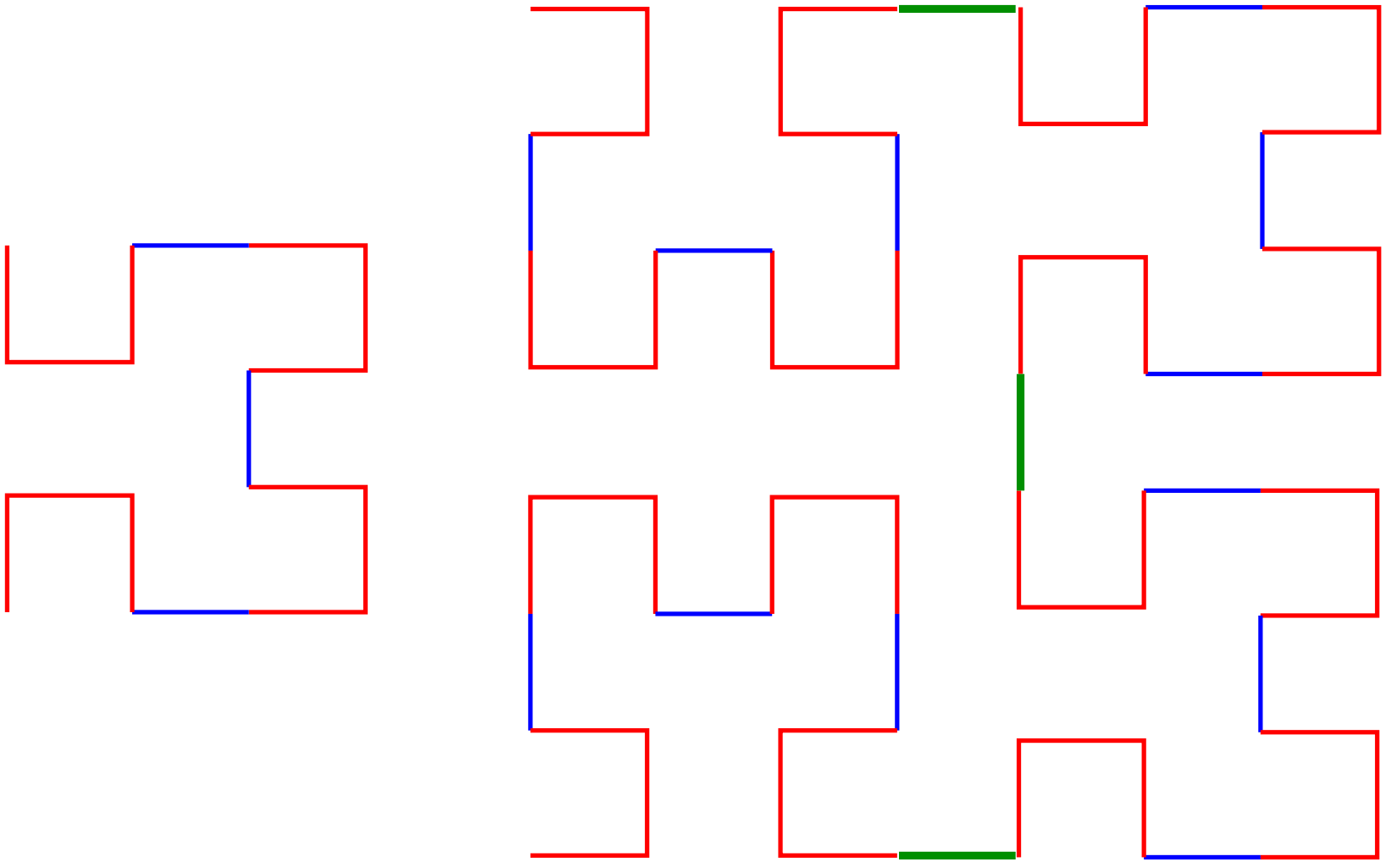


B_2 e B_3

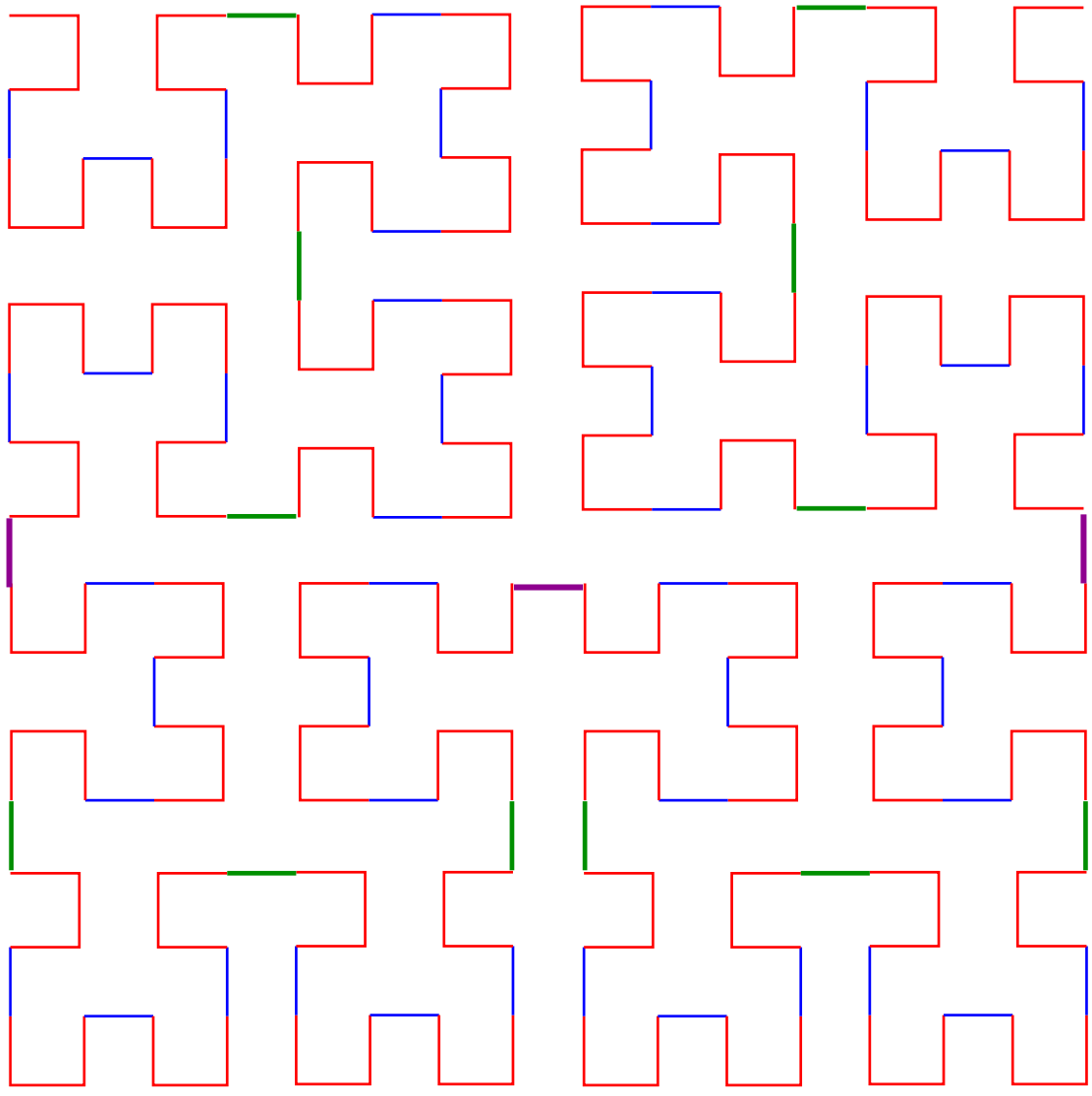




C_2 e C_3



D_4



Esquema recursivo

Assim, surge o seguinte esquema recursivo:

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

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.

linha

```
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

```
/*
 *  A
 */
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

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

C_k

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

D_k

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

Registros e Structs

PF Apêndice E

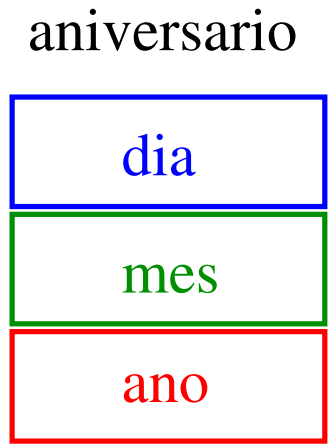
http://www.ime.usp.br/~pf/algoritmos/aulas/stru.html

Registros e structs

Um **registro** (= *record*) é uma coleção de várias variáveis, possivelmente de tipos diferentes.

Na linguagem C, registros são conhecidos como **structs**.

```
struct {
   int dia;
   int mes;
   int ano;
} aniversario ;
```

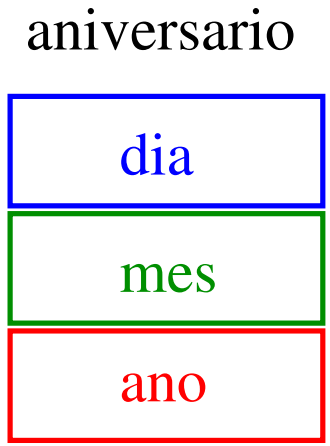


Registros e structs

É uma boa idéia dar um nome, digamos `data`, à estrutura.

Nosso exemplo ficaria melhor assim

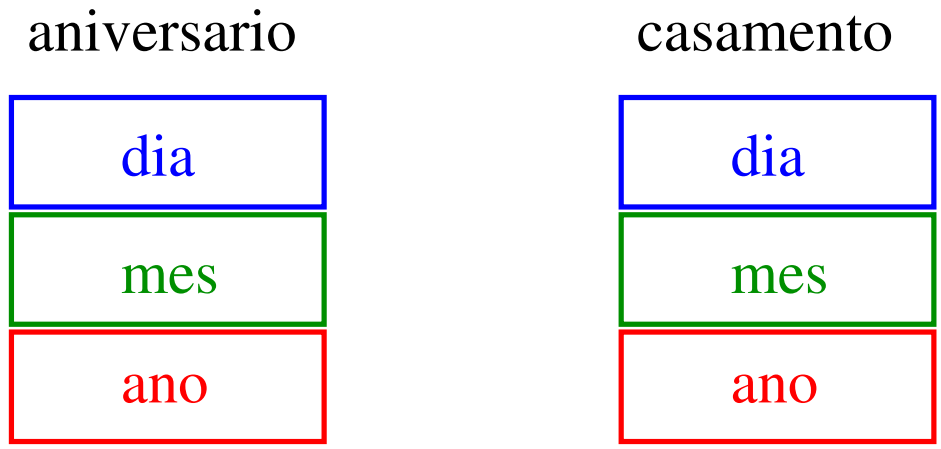
```
struct data {
    int dia;
    int mes;
    int ano;
} ;

struct data aniversario;
```



Registros e structs

Um declaração de `struct` define um tipo.

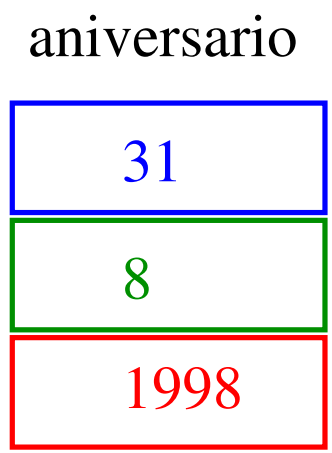
```
struct data aniversario;
struct data,casamento;
```



Registros e structs

É fácil atribuirmos valores aos campos de uma estrutura:

```
aniversario.dia = 31;
aniversario.mes = 8;
aniversario.ano = 1998;
```



Registros e structs

Para não repetir "`struct` data" o tempo todo podemos definir uma abreviatura via `typedef`:

```
struct data {
   int dia;
   int mes;
   int ano;
} ;
typedef struct data Data;
Data aniversario;
Data casamento;
```