Melhores momentos

AULA 1

Recursão

A resolução recursiva de um problema
tem tipicamente a seguinte estrutura:

```
se a instância em questão é "pequena"
   resolva-a diretamente (use força bruta
   se necessário);
senão
   reduza-a a uma instância "menor" do
   mesmo problema,
   aplique o método à instância menor e
   volte à instância original.
```

Fatorial recursivo

$$n! = \begin{cases} 1, & \text{quando } n = 0, \\ n \times (n-1)!, & \text{quando } n > 0. \end{cases}$$

```
long fatorial(long n) {
   if (n == 0) return 1;
   return n * fatorial(n-1);
}
```

Diagramas de execução

fatorial(3)

n
3

fatorial(2)

n
2

fatorial(1)

n
1

fatorial(0)

n
0

return 1

return n * fatorial(0) = 1 * 1

return n * fatorial(1) = 2 * 1 = 2

return n * fatorial(2) = 3 * 2 = 6

Fatorial iterativo

```
long fatorial(long n) {
   int i, ifat;

   ifat = 1;
   for (i = 1; /*1*/ i <= n; i++)
      ifat *= i;

   return ifat;
}
```

Em `/*1*/` vale que `ifat == (i-1)!`

Exercícios

Quantas linhas a função $\textsc{Hanoi}(\mathtt{n},\mathtt{A},\mathtt{B},\mathtt{C})$ imprime?

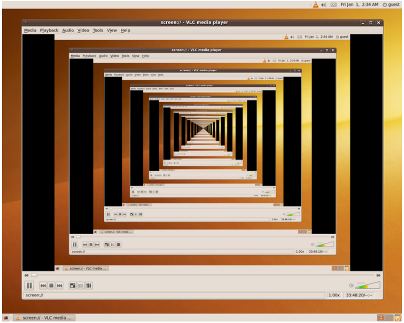
A função de Fibonacci é definida assim:

$$\mathtt{F(n)} = \begin{cases} \mathtt{n}, & \text{se } \mathtt{n} = 0 \text{ ou } \mathtt{n} = 1, \\ \mathtt{F}(\mathtt{n}-1) + \mathtt{F}(\mathtt{n}-2) & \text{se } \mathtt{n} > 1. \end{cases}$$

Escreva uma função `Fib` em linguagem C que calcule `F(n)`.
Escreva uma versão recursiva e uma iterativa da função.
Sua função recursiva é tão eficiente quanto a iterativa? Por que?

AULA 2

Mais recursão

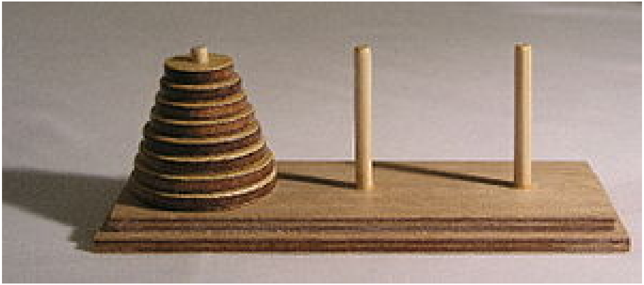


Fonte: `http://commons.wikimedia.org/`

PF 2.1, 2.2, 2.3   S 5.1

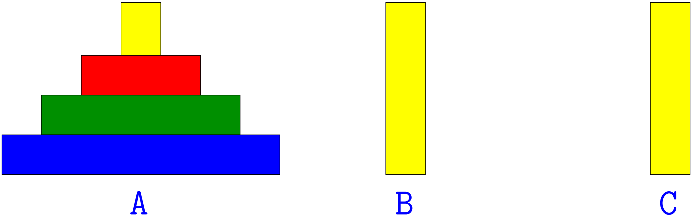
http://www.ime.usp.br/~pf/algoritmos/aulas/recu.html

Torres de Hanoi: epílogo



Fonte: http://en.wikipedia.org/

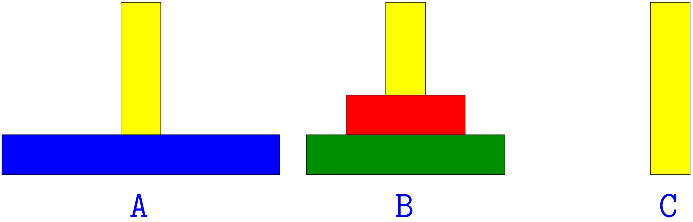
Torres de Hanoi



Desejamos transferir n discos do pino A para o pino C usando o pino B como auxiliar e repeitando as regras:

- podemos mover apenas um disco por vez;
- nunca um disco de diâmetro maior poderá ser colocado sobre um disco de diâmetro menor.

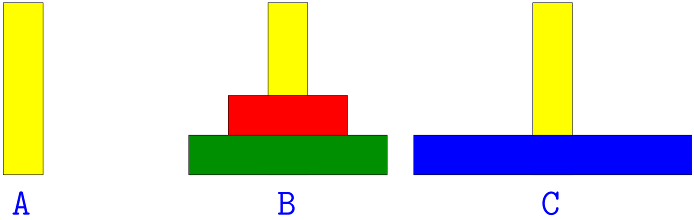
Torres de Hanoi



Desejamos transferir `n` discos do pino `A` para o pino `C` usando o pino `B` como auxiliar e repeitando as regras:

- podemos mover apenas um disco por vez;
- nunca um disco de diâmetro maior poderá ser colocado sobre um disco de diâmetro menor.

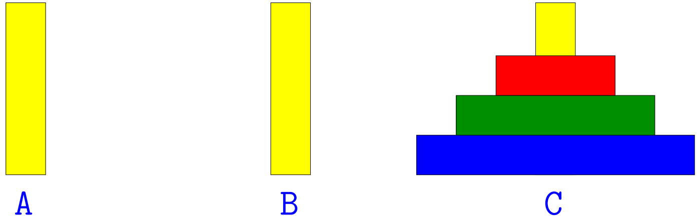
Torres de Hanoi



Desejamos transferir n discos do pino A para o pino C usando o pino B como auxiliar e repeitando as regras:

- podemos mover apenas um disco por vez;
- nunca um disco de diâmetro maior poderá ser colocado sobre um disco de diâmetro menor.

Torres de Hanoi



Desejamos transferir `n` discos do pino `A` para o pino `C` usando o pino `B` como auxiliar e repeitando as regras:

- podemos mover apenas um disco por vez;
- nunca um disco de diâmetro maior poderá ser colocado sobre um disco de diâmetro menor.

Função que resolve o problema

```
void hanoi(int n, char origem,
           char auxiliar, char destino) {
    if (n > 0) {
        hanoi(n-1, origem, destino, auxiliar);
        printf("mova disco %d de %c para %c.\n",
               n, origem, destino);
        hanoi(n-1, auxiliar, origem, destino);
    }
}
```

Quantas linhas imprime = número de movimentos feitos.

Número de movimentos

Seja $\mathtt{T}(\mathtt{n})$ o número de movimentos feitos pelo algoritmo para resolver o problema das torres de Hanoi com $\mathtt{n}$ discos.

Temos que

$$\begin{aligned} \mathtt{T}(\mathtt{0}) &= 0 \\ \mathtt{T}(\mathtt{n}) &= 2\,\mathtt{T}(\mathtt{n}-1)+1 \quad \text{para } \mathtt{n} = 1,2,3\ldots \end{aligned}$$

Relação de recorrência!

Quanto vale $\mathtt{T}(\mathtt{n})$?

Recorrência

Temos que

$$
\begin{aligned}
\mathtt{T}(\mathtt{n}) &= 2\,\mathtt{T}(\mathtt{n}-1)+1 \\
&= 2\,(2\,\mathtt{T}(\mathtt{n}-2)+1)+1 \\
&= 2\,(2\,(2\,\mathtt{T}(\mathtt{n}-3)+1)+1)+1 \\
&= 2\,(2\,(2\,(2\,\mathtt{T}(\mathtt{n}-4)+1)+1)+1)+1 \\
&= \cdots \\
&= 2\,(2\,(2\,(2\,(\cdots(2\,\mathtt{T}(0)+1))+1))+1)+1
\end{aligned}
$$

Recorrência

Logo,

$$\begin{aligned} \mathtt{T(n)} &= 2^{\mathtt{n}-1} + \cdots + 2^3 + 2^2 + 2 + 1 \\ &= 2^{\mathtt{n}} - 1\,. \end{aligned}$$

n	0	1	2	3	4	5	6	7	8	9
T(n)	0	1	3	7	15	31	63	127	255	511

Conclusões

O número de movimentos feitos pela chamada
`hanoi(n,...,...,...)` é

$$2^{\mathtt{n}} - 1\,.$$

Notemos que a função `hanoi` faz o **número mínimo** de movimentos: **não é possível** resolver o quebra-cabeça com menos movimentos.

The Tower of Hanoi Story

Taken From W.W. Rouse Ball & H.S.M. Coxeter, Mathematical Recreations and Essays, 12th edition. Univ. of Toronto Press, 1974. The De Parville account of the origen from La Nature, Paris, 1884, part I, pp. 285-286.

> *In the great temple at Benares beneath the dome that marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disk resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah. Day and night unceasingly the priest transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle which at creation God placed them, to one of the other needles, tower, temple, and Brahmins alike will crumble into dust and with a thunderclap the world will vanish . . .*

http://www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/Story_TOH.html

Enquanto isto ... os monges ...

$$\mathtt{T}(64) = 18.446.744.073.709.551.615 \approx 1{,}84 \times 10^{19}$$

Suponha que os monges façam
o movimento de 1 disco por segundo(!).

$$\begin{aligned}
18 \times 10^{19}\,\text{seg} &\approx 3{,}07 \times 10^{17}\,\text{min} \\
&\approx 5{,}11 \times 10^{15}\,\text{horas} \\
&\approx 2{,}13 \times 10^{14}\,\text{dias} \\
&\approx 5{,}83 \times 10^{11}\,\text{anos.} \\
&= \mathbf{583}\,\text{bilhões de anos.}
\end{aligned}$$

A idade da Terra é **4,54 bilhões de anos**.

The Tower of Hanoi Story

Taken From W.W. Rouse Ball & H.S.M. Coxeter, Mathematical Recreations and Essays, 12th edition. Univ. of Toronto Press, 1974. The De Parville account of the origen from La Nature, Paris, 1884, part I, pp. 285-286.

> *In the great temple at Benares beneath the dome that marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disk resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah. Day and night unceasingly the priest transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle which at creation God placed them, to one of the other needles, tower, temple, and Brahmins alike will crumble into dust and with a thunderclap the world will vanish. The number of separate transfers of single discs which the Brahmins must make to effect the transfer of the tower is two raised to the sixty-fourth power minus 1 or 18,446,744,073,709,551,615 moves. Even if the priests move one disk every second, it would take more than 500 billion years to relocate the initial tower of 64 disks.*

http://www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/Story_TOH.html

Números de Fibonacci

Fonte: http://www.geek.com/geek-cetera/

PF 2.3 S 5.2

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

Números de Fibonacci

$$\mathtt{F}_0 = 0 \qquad \mathtt{F}_1 = 1 \qquad \mathtt{F}_\mathtt{n} = \mathtt{F}_{\mathtt{n}-1} + \mathtt{F}_{\mathtt{n}-2}$$

$\mathtt{n}$	0	1	2	3	4	5	6	7	8	9
$\mathtt{F}_\mathtt{n}$	0	1	1	2	3	5	8	13	21	34

Números de Fibonacci

$$\mathtt{F}_0 = 0 \qquad \mathtt{F}_1 = 1 \qquad \mathtt{F}_\mathtt{n} = \mathtt{F}_{\mathtt{n}-1} + \mathtt{F}_{\mathtt{n}-2}$$

n	0	1	2	3	4	5	6	7	8	9
$\mathtt{F}_\mathtt{n}$	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para $\mathtt{F}_\mathtt{n}$:

```
long fibonacciR(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return fibonacciR(n-1) +
          fibonacciR(n-2);
}
```

fibonacciR(4)

```
fibonacciR(4)
  fibonacciR(3)
    fibonacciR(2)
      fibonacciR(1)
      fibonacciR(0)
    fibonacciR(1)
  fibonacciR(2)
    fibonacciR(1)
    fibonacciR(0)
fibonacci(4) = 3.
```

Fibonacci iterativo

```
long fibonacciI(int n) {
   long anterior = 0, atual = 1, proximo;
   int i;

   if (n == 0) return 0;
   if (n == 1) return 1;

   for (i = 1; i < n; i++) {
      proximo = atual + anterior;
      anterior = atual;
      atual = proximo;
   }
   return atual;
}
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 10
fibonacci(10)=55
real                        0m0.003s
user                        0m0.001s
sys                         0m0.001s

meu_prompt> time ./fibonacciR 10
fibonacci(10)=55
real                        0m0.003s
user                        0m0.001s
sys                         0m0.001s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 30
fibonacci(30) = 832040
real                        0m0.003s
user                        0m0.001s
sys                         0m0.001s

meu_prompt> time ./fibonacciR 30
fibonacci(30) = 832040
real                        0m0.009s
user                        0m0.007s
sys                         0m0.001s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 45
fibonacci(45) = 1134903170
real                        0m0.003s
user                        0m0.001s
sys                         0m0.001s

meu_prompt> time ./fibonacciR 45
fibonacci(45) = 1134903170
real                        0m8.486s
user                        0m8.459s
sys                         0m0.008s
```

fibonacciR(5)

fibonacciR resolve subproblemas muitas vezes.

```
fibonacciR(5)
  fibonacciR(4)
    fibonacciR(3)
      fibonacciR(2)
        fibonacciR(1)
        fibonacciR(0)
      fibonacciR(1)
    fibonacciR(2)
        fibonacciR(1)
        fibonacciR(0)
    fibonacciR(3)
      fibonacciR(2)
        fibonacciR(1)
        fibonacciR(0)
      fibonacciR(1)
  fibonacci(5) = 5.
```

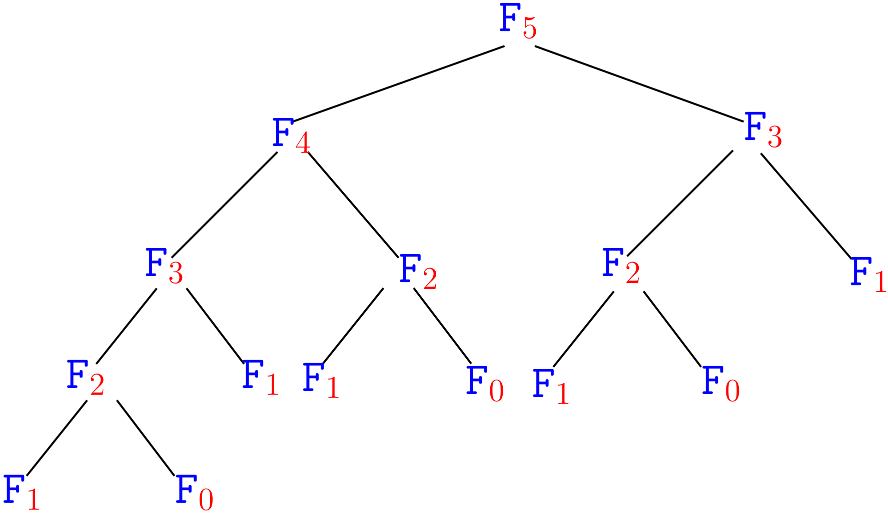
fibonacciR(8)

fibonacciR resolve subproblemas muitas vezes.

```
fibonacciR(8)
  fibonacciR(7)
    fibonacciR(6)
      fibonacciR(5)
        fibonacciR(4)
          fibonacciR(3)
            fibonacciR(2)
              fibonacciR(1)
              fibonacciR(0)
            fibonacciR(1)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
      fibonacciR(4)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
    fibonacciR(5)
      fibonacciR(4)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
      fibonacciR(3)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
        fibonacciR(1)
  fibonacciR(6)
    fibonacciR(5)
      fibonacciR(4)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
      fibonacciR(3)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
        fibonacciR(1)
    fibonacciR(4)
      fibonacciR(3)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
        fibonacciR(1)
      fibonacciR(2)
        fibonacciR(1)
        fibonacciR(0)
fibonacci(8) = 21.
```

Árvore da recursão

`fibonacciR` resolve subproblemas muitas vezes.



Desempenho de fibonacciR

Quantas chamadas recursivas faz a função fibonacciR?

```
long fibonacciR(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacciR(n-1) +
           fibonacciR(n-2);
}
```

Desempenho de fibonacciR

Quantas chamadas recursivas faz a função fibonacciR?

```
long fibonacciR(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacciR(n-1) +
           fibonacciR(n-2);
}
```

Faz o dobro do número de adições.

Desempenho de fibonacciR

Quantas chamadas recursivas faz a função fibonacciR?

```
long fibonacciR(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacciR(n-1) +
           fibonacciR(n-2);
}
```

Faz o dobro do número de adições.

Vamos calcular o número de adições feitas pela chamada fibonacciR(n).

Número de adições

Seja $\mathtt{T}(\mathtt{n})$ o número de adições
feitas pela chamada `fibonacciR(n)`.

```
long fibonacciR(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacciR(n-1) +
           fibonacciR(n-2);
}
```

Número de adições

Seja $\texttt{T}(\texttt{n})$ o número de adições
feitas pela chamada $\texttt{fibonacci}\mathsf{R}(\texttt{n})$.

linha	número de somas
1	$= 0$
2	$= 0$
3	$= \texttt{T}(\texttt{n}-1)$
4	$= \texttt{T}(\texttt{n}-2) + 1$

$$\texttt{T}(\texttt{n}) = \texttt{T}(\texttt{n}-1) + \texttt{T}(\texttt{n}-2) + 1$$

Relação de recorrência!

Recorrência

$$\mathtt{T}(0) = 0$$
$$\mathtt{T}(1) = 0$$
$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{n}-1) + \mathtt{T}(\mathtt{n}-2) + 1 \quad \text{para } \mathtt{n} = 2, 3, \ldots$$

Uma estimativa para $\mathtt{T}(\mathtt{n})$?

Recorrência

$$\mathtt{T}(0) = 0$$
$$\mathtt{T}(1) = 0$$
$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{n}-1) + \mathtt{T}(\mathtt{n}-2) + 1 \quad \text{para } \mathtt{n} = 2, 3, \ldots$$

Uma estimativa para $\mathtt{T}(\mathtt{n})$?

$\mathtt{n}$	0	1	2	3	4	5	6	7	8	9
$\mathtt{T}_\mathtt{n}$	0	0	1	2	4	7	12	20	33	54
$\mathtt{F}_\mathtt{n}$	0	1	1	2	3	5	8	13	21	34

Recorrência

$$\mathtt{T}(0) = 0$$
$$\mathtt{T}(1) = 0$$
$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{n}-1) + \mathtt{T}(\mathtt{n}-2) + 1 \quad \text{para } \mathtt{n} = 2, 3, \ldots$$

Uma estimativa para $\mathtt{T}(\mathtt{n})$?

$\mathtt{n}$	0	1	2	3	4	5	6	7	8	9
$\mathtt{T}_\mathtt{n}$	0	0	1	2	4	7	12	20	33	54
$\mathtt{F}_\mathtt{n}$	0	1	1	2	3	5	8	13	21	34

$$\mathtt{T}(\mathtt{n}) = \mathtt{F}(\mathtt{n}+1) - 1$$

Uma delimitação

$$\begin{array}{rcl} \mathtt{T}(\mathtt{n}) & = & \text{número de somas feitas por } \mathtt{fibonacciR}(\mathtt{n}) \\ & > & (3/2)^{\mathtt{n}} \text{ para } \mathtt{n} \geq 6. \end{array}$$

n	0	1	2	3	4	5	6	7	8	9
$\mathtt{T}_{\mathtt{n}}$	0	0	1	2	4	7	12	20	33	54
$(3/2)^{\mathtt{n}}$	1	1.5	2.25	3.38	5.06	7.59	11.39	17.09	25.63	38.44

Prova por indução

Prova: $T(6) = 12 > 11.40 > (3/2)^6$ e $T(7) = 20 > 18 > (3/2)^7$.
Se $n \geq 8$, então

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\overset{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\
&= (3/2 + 1)\,(3/2)^{n-2} + 1 \\
&> (5/2)\,(3/2)^{n-2} \\
&> (9/4)\,(3/2)^{n-2} \\
&= (3/2)^2 (3/2)^{n-2} \\
&= (3/2)^n .
\end{aligned}
$$

Logo, $T(n) \geq (3/2)^n$.

Consumo de tempo é **exponencial.**

Exercício

Prove uma delimitação ainda melhor, mostrando que

$$\mathtt{F}(\mathtt{n}) = \frac{\phi^{\mathtt{n}} - \hat{\phi}^{\mathtt{n}}}{\sqrt{5}} \quad \text{para } \mathtt{n} = 0, 1, 2, \ldots$$

onde

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}61803 \quad \text{e} \quad \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0{,}61803.$$

Prove que $1 + \phi = \phi^2$.

Prove que $1 + \hat{\phi} = \hat{\phi}^2$.

Conclusão

O consumo de tempo da função `fibonacciI`(`n`) é proporcional a `n`.

O consumo de tempo da função `fibonacciR` é **exponencial.**

Notação polonesa inversa

Leia o verbete da wiki:

`https://pt.wikipedia.org/wiki/Notação_polonesa_inversa`

Operandos precedem as operações:

notação infixa	notação polonesa inversa
a - b	a b -
a * b	a b *
a * b + c	a b * c +
a + b * c	a b c * +
(a + b) * c	a b + c *

Não precisa de parêntesis!

Notação polonesa inversa

Quanto valem?

(a) 8 2 + 5 /
(b) 8 2 5 + *
(c) 8 2 5 * +
(d) 2 8 2 + 5 / /

Notação polonesa inversa

Quanto valem?

(a) `8 2 + 5 /`

(b) `8 2 5 + *`

(c) `8 2 5 * +`

(d) `2 8 2 + 5 / /`

Respostas: 2 56 18 1

Linguagem PostScript

Controla uma caneta.

Linguagem PostScript

Controla uma caneta.

Algumas operações:

`moveto`:
  move a caneta para a posição dada

`lineto`:
  escreve da posição corrente até a posição dada

`stroke`:
  desenha tudo que foi escrito

Linguagem PostScript

Controla uma caneta.

Algumas operações:

`moveto`:
move a caneta para a posição dada

`lineto`:
escreve da posição corrente até a posição dada

`stroke`:
desenha tudo que foi escrito

Usa notação polonesa inversa. Vejamos um exemplo.

Desenho de um retângulo

Tamanho de uma página A4: 595×842.

Desenho de um retângulo

Tamanho de uma página A4: 595×842.

Desenha um retângulo bem próximo das bordas do papel:

```
5 5 moveto
590 5 lineto
590 837 lineto
5 837 lineto
5 5 lineto
stroke
```

Desenho de um retângulo

Tamanho de uma página A4: 595×842.

Desenha um retângulo bem próximo das bordas do papel:

```
5 5 moveto
590 5 lineto
590 837 lineto
5 837 lineto
5 5 lineto
stroke
```

Se o retângulo não estiver acertado com a página, adicione a seguinte linha no início do arquivo ps:
`<< /PageSize [595 842] >> setpagedevice`

Direção da caneta

Controla uma caneta, que tem uma direção.

Direção inicial: horizontal

Outras operações:

`rotate`:

altera a direção de um dado ângulo

`rmoveto`:

move a caneta para a posição relativa dada

`rlineto`:

escreve da posição corrente até a posição relativa dada

Desenho de um retângulo

Desenha o mesmo retângulo de outra maneira:

```
5 5 moveto
585 0 rlineto
90 rotate
832 0 rlineto
90 rotate
585 0 rlineto
90 rotate
832 0 rlineto
stroke
```

Pilha de valores

Notação polonesa inversa controla uma pilha.

Números são empilhados,
operações desempilham valores que usam.

Mais operações:

`pop`:
  joga fora o número no topo da pilha

`dup`:
  empilha uma cópia do valor no topo da pilha

Exemplo

Desenha duas figuras com lados dados na primeira linha:

```
200 400
100 100 moveto
dup 0 rlineto
90 rotate
dup 0 rlineto
90 rotate
dup 0 rlineto
90 rotate
dup 0 rlineto
90 rotate
pop
100 100 rmoveto
dup 0 rlineto
120 rotate
dup 0 rlineto
120 rotate
dup 0 rlineto
stroke
```

Exemplo

`200 400`	Pilha:	200	400		
`100 100 moveto`	Pilha:	200	400	100	100
`dup 0 rlineto`	Pilha:	200	400	400	0
`90 rotate`	Pilha:	200	400	90	
`dup 0 rlineto`	Pilha:	200	400	400	0
`90 rotate`	Pilha:	200	400	90	
`dup 0 rlineto`	Pilha:	200	400	400	0
`90 rotate`	Pilha:	200	400	90	
`dup 0 rlineto`	Pilha:	200	400	400	0
`90 rotate`	Pilha:	200	400	90	
`pop`	Pilha:	200			
`100 100 rmoveto`	Pilha:	200	100	100	
`dup 0 rlineto`	Pilha:	200	200	0	
`120 rotate`	Pilha:	200	120		
`dup 0 rlineto`	Pilha:	200	200	0	
`120 rotate`	Pilha:	200	120		
`dup 0 rlineto`	Pilha:	200	200	0	
`stroke`	Pilha:	200			

Missão para segunda-feira

1. Escrever um arquivo ps que desenhe numa página a curva de Koch de ordem 1.
2. Escrever um programa com uma função que recebe como parâmetro um inteiro `n` e imprime os comandos para imprimir a curva de Koch de ordem `n` a partir do ponto e direção em que a caneta se encontra, e usando como largura do desenho o valor que se encontra no topo da pilha do postscript. Na função `main` do seu programa, imprima os comandos iniciais e finais: coloque a caneta na posição inicial, empilhe a largura desejada do desenho, acione a sua função com o `n` que quiser, e não se esqueça do stroke no final.

Exercício

O binomial de n e k pode ser definido pela chamada regra de Pascal como:

$$\binom{n}{k} = \begin{cases} 0, & \text{quando } n = 0 \text{ e } k > 0, \\ 1, & \text{quando } n \geq 0 \text{ e } k = 0, \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{quando } n, k > 0. \end{cases}$$

Escreva uma função `binom` em linguagem C, recursiva, que calcule o binomial de n e k, baseando-se na regra de Pascal. Reflita sobre o tempo que essa função recursiva vai gastar. Pense em um jeito iterativo de implementar a mesma função, que consuma menos tempo ao executar.