

Subalgoritmos

Algoritmos e Programação de Computadores

Guilherme N. Ramos
gnramos@unb.br

2015/2



Subalgoritmos

Algoritmos iterativos permitem algo “útil” como \sqrt{n} .

```
1 while (r*r < n)
2     ++r;
3 /* ou */
4 while (abs(r*r - n) > r)
5     r = (r + (n/r)) / 2;
6 /* ou ... */
```

É preciso saber *como realizar* a computação, mas é mais interessante saber *como conseguir* o resultado.

Subalgoritmos



Abstrações permitem separar os detalhes da implementação dos da utilização da computação.

Subalgoritmos

A modularização do algoritmo facilita:

- planejamento/implementação da solução;
- composição/compreensão do código;
- reuso de um mesmo módulo em diversas aplicações.

Implementação

```
1 if (x < y)
2     z = x;
3 else
4     z = y;
5
6 while (abs(r*r - n) > r)
7     r = (r + (n/r)) / 2;
```

Abstração

```
1 z = min(x, y);
2
3 r = raiz2(n);
```

Funções

Funções/procedimentos são a implementação de subalgoritmos:

- são o primeiro passo na organização do programa;
- dividem um algoritmo em subalgoritmos menores (mais fáceis);
- podem ser implementadas por programadores diferentes;
- podem ser utilizadas em sistemas diferentes;

A função é chamada pelo `identificador`, recebendo argumentos para processar (ou não), e retornando um `resultado` (ou não).

```
1 desligue_o_computador()  
2 data ← que_dia_e_hoje()  
3 resultado ← eleva_ao_cubo(2)
```

Escopos

O *escopo* é um formalismo que associa o par $\langle \text{escopo}, \text{identificador} \rangle$ ao valor armazenado em memória.

Escopo Local

O identificador tem significado apenas no bloco em que foi declarado, e sobrepõe-se a outro identificador igual (se houver).

Escopo Global

O identificador tem significado em qualquer escopo (a menos que sobreposto por um identificador idêntico em um escopo local).

Recursividade

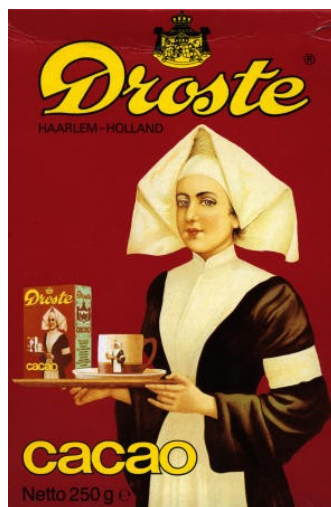
s. f.: veja *Recursividade*

Recursão¹

Termo usado de maneira mais geral para descrever o processo de repetição de um objeto de um jeito similar ao que já fora mostrado.

Em matemática/programação, uma *função recursiva* é aquela que chama a si mesma.

$$n! = n(n-1)!$$



¹“Para entender recursão, você precisa entender recursão.” - David Hunter

Recursividade

```
1 Função inteiro Fatorial(inteiro n)  
2 Início  
3  
4     Retorne( Fatorial( ))  
5 Fim
```

A primeira coisa a ser feita quando implementar com uma função recursiva é o *critério de parada*.

```
9-fatorial.c  
1 int fatorial_r(int n) {  
2     if(n < 2)  
3         return 1;  
4  
5     return n*fatorial_r(n-1);  
}
```

Módulos

Linguagens de programação possuem uma forma de incluir conteúdo: modularização e reutilização

Bibliotecas de código:

- simplificam a referência
- simplificam a manutenção
- garantem que todos usam as mesmas instruções

```
1 #include <stdio.h>
2 #include <math.h>
3 /* ... */
```

Duplicação?

apc_subalgoritmos.h.

“Sugestões”

Ao codificar as funções, tente mantê-las:

- curtas!
- organizadas (blocos e indentação);
- fazendo **uma** coisa *direito*;
- com nomes adequadamente descritivos;
- com poucos argumentos;
- sem efeitos colaterais.

