

Subalgoritmos

Algoritmos e Programação de Computadores

Guilherme N. Ramos
gnramos@unb.br

2018/2



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)
```

Newton-Raphson

Análise Numérica

Ramo da matemática que estuda algoritmos que convergem para resultados [matematicamente válidos] de problemas [matemáticos].

O ideal é chegar ao valor mais próximo viável com o mínimo de iterações.

Método de Newton-Raphson

Algoritmo para aproximar os valores das raízes de uma função.

Dado um polinômio qualquer

$$f(x) = a_n x^n + a_{n-1} x^{n-1} \dots a_1 x + a_0$$

deseja-se a raiz r tal que $f(r) = 0$.

Newton-Raphson

Newton provou que uma boa aproximação da raiz é $\frac{f}{f'}$.

Por exemplo, a raiz quadrada de 9 seria a raiz do polinômio $f(r) = r^2 - 9$.

Portanto

$$r \approx \frac{f}{f'} = \frac{r^2 - 9}{2r}$$

Assim, partindo de um valor r inicial, pode-se realizar sucessivas iterações para aproximar o valor real da raiz do polinômio.

r	$r^2 - 9$	$2r$	$\frac{r^2 - 9}{2r}$

main

As entradas e saídas das funções podem (e devem) ser concatenadas entre si, mas para uso correto, é preciso saber como se comunicar com elas (E/S) e o que elas fazem - mas não [necessariamente] como elas o fazem.

`main` é a função de entrada dos programas em C e retorna um valor inteiro que é lido pelo sistema operacional. Historicamente:

- 0 Não houve erro (EXIT_SUCCESS)
- $\neq 0$ Houve erro (geralmente o número indica *qual* erro.)

main

01-main.c

```
1 /**      @file: 01-main.c
2 *      @author: Guilherme N. Ramos (gnramos@unb.br)
3 * @disciplina: Algoritmos e Programação de Computadores
4 *
5 * Exemplo de valor de retorno arbitrário. */
6
7 #include <stdio.h>
8
9 #define ERRO_DE_LOGICA 8
10
11 int main() {
12     /* código que faz algo errado. */
13     return ERRO_DE_LOGICA;
14 }
```

main

02-main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     int resultado;
6
7     /*****
8      * ATENÇÃO! *
9      *****/
10    resultado = system("gcc -Wall -ansi ../01-main/01-main.c
11    -o teste");
12    printf("Resultado de gcc: %d \n", WEXITSTATUS(resultado));
13
14    if(resultado == EXIT_SUCCESS) {
15        resultado = system("./teste");
16        printf("Resultado da execução: %d\n",
17        WEXITSTATUS(resultado));
18    }
19    return EXIT_SUCCESS;
20 }
```

main

01-main.py

```
1 #  -*- coding: utf-8 -*-
2 #      @package: 01-main.py
3 #      @author: Guilherme N. Ramos (gnramos@unb.br)
4 # @disciplina: Algoritmos e Programação de Computadores
5 #
6 # Exemplo de valor de retorno arbitrário.
7
8
9 import sys
10
11 ERRO_DE_LOGICA = 8
12
13
14 # código que faz algo errado
15 sys.exit(ERRO_DE_LOGICA) # a função sys.exit retorna o
16                          # resultado dado ao sistema
17                          # operacional
```

main

02-main.py

```
1 import subprocess
2
3 if __name__ == '__main__':
4     # ****
5     # * ATENÇÃO! *
6     # ****
7     comando = 'python ../01-main/01-main.py'
8     resultado = subprocess.call(comando, shell=True)
9     print('Resultado da execução:', resultado)
```

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*

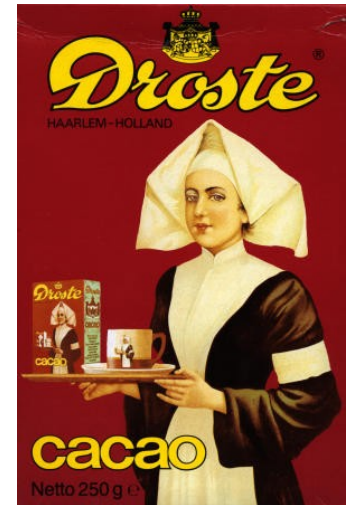
Recursão¹

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
6
```

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

09-fatorial/fatorial.c

```
1 int fatorial_r(int n) {
2     if (n < 1)
3         return 1;
4
5     return n * fatorial_r(n - 1);
6 }
```

Máximo Divisor Comum

O *MDC* entre dois ou mais números inteiros (diferentes de 0) é o maior número inteiro que é fator de tais números.

$$\text{mdc}(12, 18) = 6$$

$$\text{mdc}(54, 24) = 6$$

$$\text{mdc}(10, 15) = 5$$

$$\text{mdc}(10, 20) = 10$$

$$\text{mdc}(10, 25) = 5$$

$$\text{mdc}(10, 30) = 10$$

Dois números inteiros a e b são primos entre si, se e somente se

$$\text{mdc}(a, b) = 1.$$

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 /* ... */
4
```

Duplicação?

apc.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.