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

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

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 uma 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

```c
/**      @file: 01-main.c
 *     @author: Guilherme N. Ramos (gnramos@unb.br)
 * @disciplina: Algoritmos e Programação de Computadores
 *
 * Exemplo de valor de retorno arbitrário. */

#include <stdio.h>

#define ERRO_DE_LOGICA 8

int main() {
    /* código que faz algo errado. */
    return ERRO_DE_LOGICA;
}
```


main

02-main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
  int resultado;

  /************
   * ATENÇÃO! *
   ************/
  resultado = system("gcc -Wall -ansi ../01-main/01-main.c
-o teste");
  printf("Resultado de gcc: %d \n", WEXITSTATUS(resultado));

  if(resultado == EXIT_SUCCESS) {
    resultado = system("./teste");
    printf("Resultado da execução: %d\n",
WEXITSTATUS(resultado));
  }

  return EXIT_SUCCESS;
}
```


main

01-main.py

```python
#  -*- coding: utf-8 -*-
#    @package: 01-main.py
#     @author: Guilherme N. Ramos (gnramos@unb.br)
# @disciplina: Algoritmos e Programação de Computadores
#
# Exemplo de valor de retorno arbitrário.


import sys

ERRO_DE_LOGICA = 8


# código que faz algo errado
sys.exit(ERRO_DE_LOGICA)  # a função sys.exit retorna o
                          # resultado dado ao sistema
                          # operacional
```


main

02-main.py

```python
import subprocess

if __name__ == '__main__':
    # ************
    # * ATENÇÃO! *
    # ************
    comando = 'python ../01-main/01-main.py'
    resultado = subprocess.call(comando, shell=True)
    print('Resultado da execução:', resultado)
```

Escopos

O *escopo* é um formalismo que associa o par $<$ *escopo*, *identificador* $>$ 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

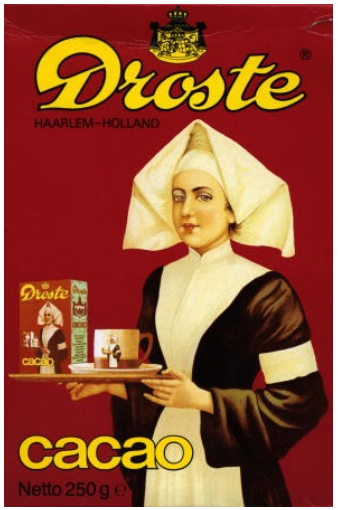
s. f.: veja *Recursividade*

Recursão[1]

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.

`mdc(12,18)` = 6

`mdc(54,24)` = 6

`mdc(10,15)` = 5

`mdc(10,20)` = 10

`mdc(10,25)` = 5

`mdc(10,30)` = 10

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