Algoritmos Computacionais

Professor: Rodrigo Rocha

Aula: 8

FAVIP | DeVry Brasil

Agnda

- Decomposição de Problemas
- Construção de Módulos
- Parametrização de Módulos
- Tipos de Módulos

Decomposição de problemas

- Complexidade = **Variedade**
  - Um problema matemático é tão mais complexo quanto maior for a quantidade de variáveis a serem tratadas
  - Um problema algorítmico é tão mais complexo quanto maior for a quantidade de situações diferentes que precisam ser tratadas
- Como reduzir a Complexidade?
  - Reduzindo a Variedade
- Como reduzir a Variedade?
  - Dividindo problemas maiores em problemas pequenos
  - A divisão precisa acontecer de forma metódica

Decomposição de problemas

- Refinamentos Sucessivos
  - É uma técnica que orienta o processo de divisão de problemas. Funciona assim:

1. Divida o problema em suas partes principais
2. Analise a divisão obtida para garantir coerência
3. Se alguma parte ainda estiver complexa, voltar para 1
4. Analise o resultado para garantir entendimento e coerência

  - Essa técnica é comumente chamada de Top-Dowm

Decomposição de problemas

- Problema Exemplo: Cartão Ponto
  - A partir dos horários de entrada e saída de um funcionário, calcular o total de horas trabalhadas, o total de atraso, a média diária de horas trabalhadas e de atraso

Construção de Módulos

- Cada divisão obtida com a técnica de Refinamentos Sucessivos pode ser convertida numa parte do Algoritmo

- Cada uma dessas partes são conhecidas como **Módulos** ou **Subalgoritmos**

- Declaração:

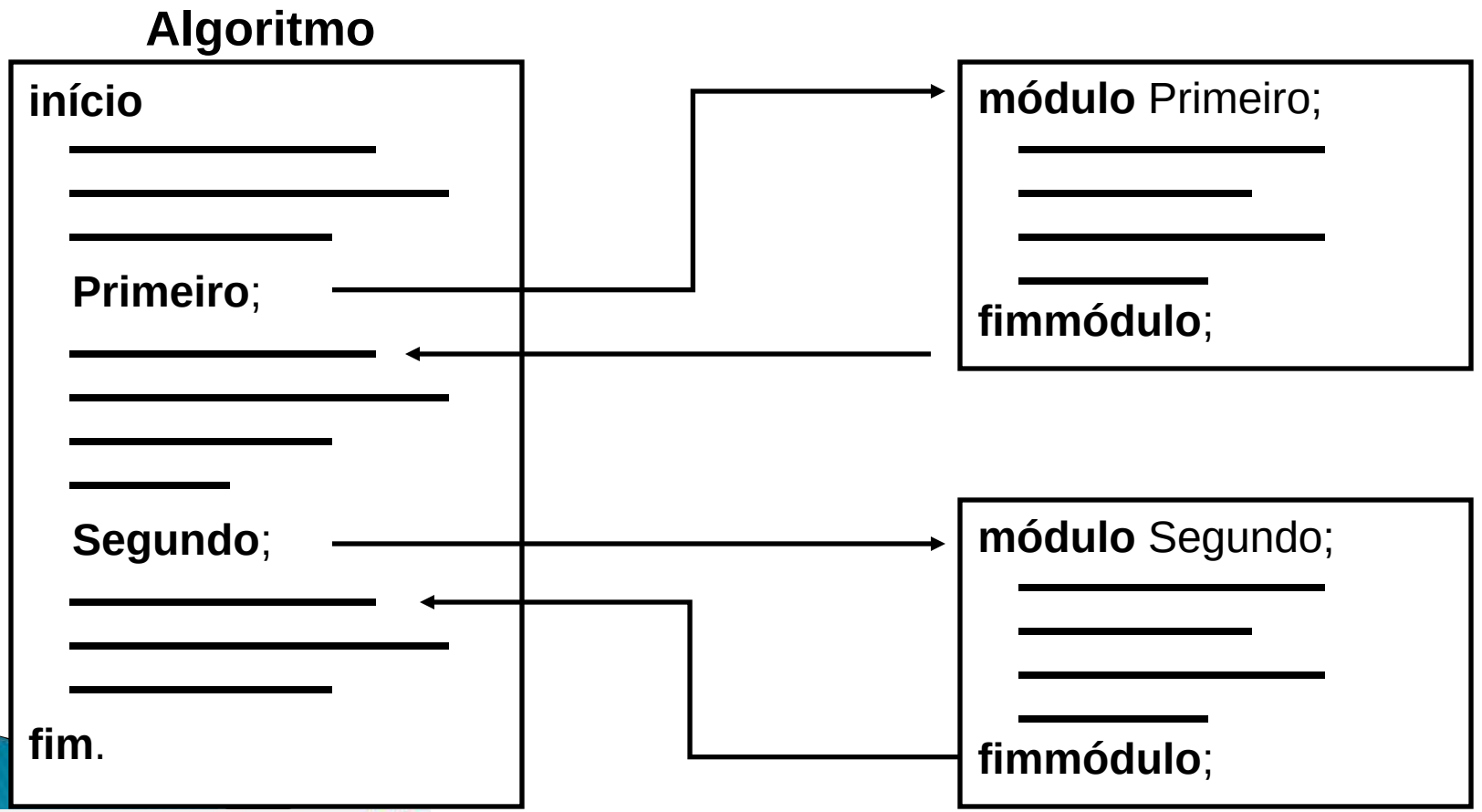
  módulo Entrada
  comando1;
  comando2;
  ...
  fimmódulo;
  Onde :
  - Entrada : Identificador do módulo que está sendo construído
  - comando1, comando2 : Algoritmo do módulo

Construção de Módulos

- Manipulação:
  - Um módulo pode ser acionado de qualquer ponto do algoritmo principal ou de outro módulo
  - O acionamento de um módulo também é conhecido por **chamada** ou **ativação** do módulo
  - Quando ocorre uma chamada o fluxo de execução, passa para o módulo chamado
  - Quando se conclui a execução do módulo chamado, o controle retorna para o módulo chamador

Construção de Módulos

- Ativação de Módulos:



Construção de Módulos

- Vantagens:
  - Clareza e legibilidade no algoritmo
  - Construção independente
  - Testes individualizados
  - Simplificação da manutenção
  - Reaproveitamento de algoritmos

Parametrização de Módulos

- É possível tornar um módulo mais genérico e portanto mais reutilizável
- Isso pode ser feito através de **Parâmetros**
- P.ex.: um módulo que imprime a tabuada do 7 tem alguma aplicabilidade. Porém, se o módulo fosse capaz de imprimir qualquer tabuada, seria mais genérico.
- Então, bastaria ativar o módulo indicando por **parâmetro** qual a tabuada desejada.

Parametrização de Módulos

- Parâmetros são os valores que um módulo pode receber do chamador antes de iniciar sua execução
- A ativação do módulo precisa indicar os argumentos (valores constantes ou variáveis) que serão passados para o módulo
- A passagem dos valores ocorre a partir da correspondência (ordem) argumento x parâmetro

Parametrização de Módulos

Algoritmo 6.7 – Módulo Troca

```
módulo Troca (inteiro: X, Y)
        inteiro: aux;
        aux <- X;
        X <- Y;
        Y <- aux;
fimmódulo;
```

Se quiséssemos trocar o valor de duas variáveis quaisquer em dado algoritmo, poderíamos fazer:

```
a <- 7;
b <- 15;
Troca (a, b);
escreva (a, b);
```

- O valor de a (7) é transferido para seu respectivo parâmetro : X, enquanto b é transferido para Y

Quando o módulo termina, ocorre o inverso

Tipos de Módulos

- Podemos classificar os módulos quanto ao seu objetivo principal em dois contextos: Ação e Resultado

- Contexto de Ação
  - Quando o módulo simplesmente executa coisas

- Contexto de Resultado
  - Quando o módulo calcula coisas e retorna um resultado

Tipos de Módulos

- Contexto de Ação

Algoritmo 6.9 – Inverte Vetor

```
Modulo Inverte (VET: VI)
    inteiro: i, aux;
    para i de 1 até 10 faça
      aux ← VI [ i ];
      VI [ i ] ← VI [ 11 - i ];
      VI [ 11 - i ] ← aux;
    fimpara;
  fimmódulo;
```

Algoritmo 6.9 – versão 2

```
módulo Inverte (VET: VI)
  inteiro: i, aux;
  para i de 1 até 10 faça
    Troca (VI [ i ], VI [ 11 - i ]);
  fimpara;
fimmódulo;
```

Tipos de Módulos

- Contexto de Resultado
  - Para que um módulo seja de Resultado, é preciso retornar para o chamador o resultado do cálculo
  - Isso pode ser feito com o comando **retorne**
    - Ex : **retorne** (5);
    - **retorne** (a);

Algoritmo 6.12 – Módulo Par

```
módulo Par (inteiro: N)
   se ((N mod 2) = 0) entao
      retorne ( V );
   senao
      retorne ( F );
   fimse;
fimmódulo;
```

Perguntas?