Algoritmos e lógica de programação: Algoritmos modularizados e com recursão

Resumo

Referências Bibliográficas

- ARAÚJO, Sandro de. **Lógica de Programação e Algoritmos [recurso eletrônico].** Curitiba. Contentus. 2020.
- FORBELLONE, André Luiz Villar e EBERSPÄCHER, Henri Frederico. **Lógica de Programação: a construção de algoritmos e estruturas de dados.** 3a Edição. São Paulo. Prentice Hall. 2005.

Quando desenvolvemos algoritmos, precisamos nos preocupar em como desenvolvê-los de forma que sejam facilmente entendidos. Para isso, podemos, em vez de criar um único algoritmo, quebrá-lo em partes e simplificar o problema, para depois, quando reunidos, resolvem o problema maior.

Para isso, vamos estudar os **módulos** em algoritmos, que são os pedaços menores do algoritmo maior, que resolve problemas menores para que depois o problema maior seja resolvido. Podemos escrever quatro tipos de módulos: o procedimento com parâmetro, o procedimento sem parâmetro, a função com parâmetro e a função sem parâmetro.

Vamos estudar também a forma recursiva de se desenvolver um módulo. A recursão substitui as estruturas de repetição em problemas em que a quantidade de repetições é determinada de acordo com outras condições. A recursão minimiza as linhas de comandos, porém, exige mais raciocínio lógico e matemático.

Módulos em algoritmos

- **Procedimento sem parâmetro**: é a estrutura que resolve um problema menor sem receber qualquer informação de outro lugar do algoritmo e não retorna valor para outro lugar do algoritmo.

Este módulo é como um algoritmo que resolve um problema simples, como abaixo:

```
//módulo procedimento sem parâmetro
<nome do módulo> ()
  início
    <comandos>;
  fimmódulo;
```

Exemplo:

```
// módulo procedimento chamado Dados que não recebe argumentos no parâmetro
// e não retorna qualquer valor
Dados ()
  início
    Declarar
      disciplina alfanumérico;

    disciplina <- "Algoritmos e Lógica de Programação";
    escreva ("nome da disciplina => " , disciplina);
```

fimmódulo;

- **Função sem parâmetro**: é a estrutura que resolve um problema menor sem receber qualquer informação de outro lugar do algoritmo e retorna algum valor para outro lugar do algoritmo.

Este módulo é como um algoritmo que resolve um problema simples e que retorna um valor, como abaixo:

```
//módulo função sem parâmetro
<tipo de retorno> <nome do módulo> ()
  início
    <comandos>;
    retornar <valor de retorno>;
  fimmódulo;
```

Exemplo:

```
// módulo função chamado Dados que não recebe argumentos no parâmetro
// e retorna algum valor do tipo alfanumérico
alfanumérico Dados ()
  início
    Declarar
      disciplina alfanumérico;
     disciplina <- "Algoritmos e Lógica de Programação";
    retornar ("nome da disciplina => " + disciplina);
  fimmódulo;
```

- **Procedimento com parâmetro:** é a estrutura que resolve um problema menor recebendo alguma(s) informação(ões) de outro lugar do algoritmo e não retorna valor para outro lugar do algoritmo.

Este módulo é como um algoritmo que resolve um problema simples, como abaixo:

```
//módulo procedimento com parâmetro
<nome do módulo> (<tipo var> <nome var> , ... , <tipo var> <nome var>)
  início
    <comandos>;
  fimmódulo;
//disciplina <- "Algoritmos e Lógica de Programação“;
```

Exemplo:

```
// módulo procedimento chamado Dados que
// recebe a variável disciplina do tipo alfanumérico como argumento no parâmetro
// e não retorna qualquer valor
Dados (disciplina alfanumérico)
  início

    escreva ("nome da disciplina => " , disciplina);
  fimmódulo;
```

- **Função com parâmetro:** é a estrutura que resolve um problema menor recebendo alguma(s) informação(ões) de outro lugar do algoritmo e retorna algum valor para outro lugar do algoritmo.

Este módulo é como um algoritmo que resolve um problema simples e retorna algum valor para outro lugar do algoritmo, como abaixo:

```
//módulo função com parâmetro
<tipo retorno> <nome módulo> (<tipo var> <nome var> , ... , <tipo var> <nome var>)
  início
    <comandos>;
    retornar <valor de retorno>;
  fimmódulo;
//disciplina <- "Algoritmos e Lógica de Programação“;
```

Exemplo:

```
// módulo função chamado Dados que
// recebe a variável disciplina do tipo alfanumérico como argumento no parâmetro
// e retorna algum valor do tipo alfanumérico
alfanumérico Dados (disciplina alfanumérico)
  início
    retornar ("nome da disciplina => " + disciplina);
  fimmódulo;
```

Agora, precisamos realizar a chamada adequada de cada um dos módulos criados. Os módulos só são executados se forem chamados pelo módulo principal ou por outro módulo.

```
Declarar
  Disc, Nova <- ”Algoritmos” alfanumérico;
//chamada do procedimento sem parâmetro
Dados ( );
//chamado da função sem parâmetro
Disc <- Dados( );
//chamada do procedimento com parâmetro
Dados (Nova);
//chamado da função com parâmetro
Disc <- Dados(Nova);
```

Estrutura de módulos em algoritmos

Para praticar as estruturas de modularização, vamos desenvolver um algoritmo que recebe dois valores numérico_inteiro e o símbolo da operação +, -, * ou /, calcula e mostra, respectivamente, o resultado da adição, da subtração, da multiplicação e da divisão dos dois inteiros.

Vamos entender a situação-problema: a entrada de dados é dois números inteiros. O processamento é o cálculo da adição, multiplicação, subtração e divisão dos dois inteiros, que podem ser realizados em módulos. Quebraremos o problema maior em quatro menores, um módulo cada para calcular a adição, subtração, multiplicação e divisão.

O módulo Mais é um módulo função que recebe no parâmetro dois inteiros, n1 e n2 como argumentos, retornando um inteiro. Este módulo calcula a soma do conteúdo das variáveis n1 e n2 e retorna o resultado da soma.

```
/* módulo função Mais que recebe dois valores inteiros e retorna um valor inteiro */
numérico_inteiro Mais (n1 numérico_inteiro , n2 numérico_inteiro)
  início
    Declarar
      res numérico_inteiro;
    res <- n1 + n2; // processamento de dados
    retornar res; // retorno do módulo
  fimmódulo;
```

O módulo Menos é um módulo procedimento que recebe no parâmetro dois inteiros, n1 e n2 como argumentos. Este módulo calcula a diferença do conteúdo das variáveis n1 e n2 e mostra o resultado da diferença.

```
// módulo procedimento Menos que recebe dois valores inteiros
Menos (n1 numérico_inteiro, n2 numérico_inteiro)
  início
    Declarar
       res numérico_inteiro;
    res <- n1 - n2; // processamento de dados
    // saída de resultados
    escrever ("A diferença de " , n1 , " com " , n2 , " é " , res);
  fimmódulo;
```

O módulo Vezes é um módulo procedimento que recebe no parâmetro dois inteiros, n1 e n2 como argumentos. Este módulo calcula o produto do conteúdo das variáveis n1 e n2 e mostra o resultado do produto.

```
// módulo procedimento Vezes que recebe dois valores inteiros
Vezes (n1 numérico_inteiro, n2 numérico_inteiro)
  início
    Declarar
      res numérico_inteiro;
    res <- n1 * n2; // processamento de dados
            // saída de resultados
    escrever ("O produto de " , n1 , " com " , n2 , " é " , res);
  fimmódulo;
```

O módulo Dividido é um módulo função que recebe no parâmetro dois inteiros, n1 e n2 como argumentos, retornando um real. Este módulo calcula o resultado da divisão do conteúdo das variáveis n1 e n2 e retorna o resultado do cálculo da divisão.

```
/* módulo função Dividido que recebe dois valores inteiros e retorna um valor real */
numérico_real Dividido (n1 numérico_inteiro, n2 numérico_inteiro)
  início
    Declarar
      res numérico_real;
```

```
    res <- n1 / n2; // processamento de dados
    retornar res; // retorno do módulo
  fimmódulo;
```

O algoritmo principal Calculo declara as variáveis, faz a entrada de dados de dois inteiros pelo usuário, faz a chamada adequada de cada um dos módulos e apresenta o resultado dos cálculos da funções.

```
Algoritmo Calculo
  início_algoritmo
    Declarar
      div numérico_real;
      soma, num1, num2 numérico_inteiro;
      oper alfanumérico;
    escrever ("Digite dois inteiros e a operação");
    ler (num1, num2, oper);
    soma <- Mais(num1 , num2);
    escrever ("A soma de " , num1 , " com " , num2 , " é " , soma);
    Menos (num1 , num2);
    Vezes (num1 , num2);
    div <- Dividido (num1 , num2);
    escrever ("A divisão de " , num1 , " com " , num2 , " é " , div);
  fim_algoritmo.
```

Recursão em Algoritmos

Para criar algoritmos recursivos, utilizamos funções e procedimentos. Quando desenvolvemos um algoritmo recursivo com o módulo função, dizemos que o método recursivo é com cauda. Quando desenvolvemos um algoritmo recursivo com o módulo procedimento, dizemos que o método recursivo é sem cauda.

Um método é recursivo quando realizamos a chamada deste método dentro dele mesmo, podendo ser função ou procedimento. Um método recursivo substitui uma estrutura de repetição, pois sua chamada dentro dele mesmo é, na verdade, uma chamada repetitiva do próprio módulo.

Quando desenvolvemos algoritmos recursivos, é importante se atentar para três regras importantes da recursão, são elas:

1) Saber quando parar.
   A primeira regra garante a parada do método recursivo.
2) Decidir como fazer a próxima ação.
   Segunda regra garante a continuidade da recursão.
3) Quebrar uma jornada recursiva em um passo mais uma jornada recursiva menor.
   Terceira regra garante um passo menor da recursão.

Por exemplo, vamos calcular o fatorial de um número n natural:

para n = 0, 0! = 1
para n = 1, 1! = 1
para n =2, 2! = 1 * 2 = 1! * 2 = 2
para n = 3, 3! = 1 * 2 * 3 = 2! * 3 = 6
para n=4, 4! = 1 * 2 * 3 * 4 = 3! * 4 = 24

para n = N, N! = 1 * 2 * 3 * .... * (N-2) * (N-1) * N = (N-1)! * N
Pensando na forma de função:
Para n>=0, n inteiro
- se n = 0 então, o valor de n! é 1
- caso contrário, n! = 1 * 2 * 3 * ... * (n-2) * (n-1) * n = (n-1)! * n

Pensando na forma computacional:
fat(n) = 1 se n = 0
fat(n) = n * fat(n-1), caso contrário.

Escrevendo a função recursiva para o fatorial de um número, considerando o pensamento computacional, temos:

```
// módulo função recursivo
numérico_inteiro fat (numérico inteiro n)
  início
    Declarar
      f númerico_inteiro;
    se (n = 0)
      então
        retornar 1; // regra 1
      senão
        f <- n * fat(n-1); // regra 2 e 3
        retornar f;
    fimse;
  fim_módulo;
```

Por exemplo, para n=5, na chamada recursiva, temos:
f <- fat(5)
f <- fat(4) * 5
f <- fat(3) * 4 * 5
f <- fat(2) * 3 * 4 * 5
f <- fat(1) * 2 * 3 * 4 * 5
f <- fat(0) * 1 * 2 * 3 * 4 * 5
f <- 1 * 1 * 2 * 3 * 4 * 5
f <- 1 * 2 * 3 * 4 * 5
f <- 2 * 3 * 4 * 5
f <- 6 * 4 * 5
f <- 24 * 5
f <- 120

O resultado da chamada recursiva da função que calcula o fatorial de 5, é igual a 120.

Estrutura de Recursão em Algoritmos

Agora, vamos exercitar a estrutura recursiva desenvolvendo um algoritmo que recebe do usuário um número inteiro e calcula o fatorial recursivo desse número com procedimento e com função.

Primeiro, vamos criar o módulo função recursivo, recebendo um inteiro como argumento no parâmetro e retornando um inteiro. O resultado parcial do cálculo do fatorial vai ficando na cauda do módulo função, a cada vez que este módulo é chamado na regra 2 e 3.

```
// módulo função recursivo
numérico_inteiro fat (numérico_inteiro n)
  início
    Declarar
      f númerico_inteiro;
    se (n = 0)
      então
        retornar 1; // regra 1
      senão
        f <- n * fat(n-1); // regra 2 e 3
        retornar f;
    fimse;
  fim_módulo;
```

Agora, vamos criar um módulo procedimento recebendo dois inteiros como parâmetro. Perceba que como não devolvemos o cálculo parcial do fatorial do número, o resultado parcial vai sendo armazenado e encaminhado recursivamente no argumento f, no parâmetro do módulo procedimento fatP.

```
// módulo procedimento recursivo
fatP (numérico_inteiro n, numérico_inteiro f)
  início
    Declarar

    se (n = 0 ou n = 1)
      então
        escrever(“O fatorial com procedimento é “ , f); // regra 1
      senão
        fat(n-1, f*n); // regra 2 e 3
    fimse;
fim_módulo;
```

No algoritmo principal Fatorial, recebe-se a entrada de dados do usuário, realiza-se a chamada adequada, tanto da função quanto do procedimento, que calculam os fatoriais do número fornecido pelo usuário e os resultados dos fatoriais calculados são exibidos.

```
Algoritmo Fatorial
  início
    Declarar
      f, nro numérico_inteiro;
    escrever("Digite um valor natural para calcular o fatorial");
    ler(nro);
    f <- fat(nro);
    escrever("O fatorial de " , nro , " é " , f , " função.");
    fatP(nro, 1);
  fim_algoritmo.
```

Exercícios

1. Em algoritmos, podemos escrever os módulos, que são pedaços menores do algoritmo maior. Sobre o módulo procedimento sem parâmetro, pode-se considerar CORRETA a afirmação:

a) É um módulo que não recebe parâmetro e não retorna qualquer valor para quem o chamou.
b) É um módulo que recebe argumentos no parâmetro e não retorna qualquer valor para quem o chamou.
c) É um módulo que não recebe parâmetro e retorna algum valor para quem o chamou.
d) É um módulo que recebe argumentos no parâmetro e retorna algum valor para quem o chamou.
e) É um módulo recursivo que retorna vários valores.

2. Em algoritmos, podemos escrever os módulos, que são pedaços menores do algoritmo maior. Sobre o módulo procedimento com parâmetro, pode-se considerar CORRETA a afirmação:

a) É um módulo que não recebe parâmetro e não retorna qualquer valor para quem o chamou.
b) É um módulo que recebe argumentos no parâmetro e não retorna qualquer valor para quem o chamou.
c) É um módulo que não recebe parâmetro e retorna algum valor para quem o chamou.
d) É um módulo que recebe argumentos no parâmetro e retorna algum valor para quem o chamou.
e) É um módulo recursivo que retorna vários valores.

3. Em algoritmos, podemos escrever os módulos, que são pedaços menores do algoritmo maior. Sobre o módulo função sem parâmetro, pode-se considerar CORRETA a afirmação:

a) É um módulo que não recebe parâmetro e não retorna qualquer valor para quem o chamou.
b) É um módulo que recebe argumentos no parâmetro e não retorna qualquer valor para quem o chamou.
c) É um módulo que não recebe parâmetro e retorna algum valor para quem o chamou.
d) É um módulo que recebe argumentos no parâmetro e retorna algum valor para quem o chamou.
e) É um módulo recursivo que retorna vários valores.

4. Em algoritmos, podemos escrever os módulos, que são pedaços menores do algoritmo maior. Sobre o módulo função com parâmetro, pode-se considerar CORRETA a afirmação:

a) É um módulo que não recebe parâmetro e não retorna qualquer valor para quem o chamou.
b) É um módulo que recebe argumentos no parâmetro e não retorna qualquer valor para quem o chamou.
c) É um módulo que não recebe parâmetro e retorna algum valor para quem o chamou.
d) É um módulo que recebe argumentos no parâmetro e retorna algum valor para quem o chamou.
e) É um módulo recursivo que retorna vários valores.

5. Avalie as seguintes afirmações considerando os métodos recursivos:

I. Para criar algoritmos recursivos, utilizamos funções e procedimentos.
II. Um método recursivo substitui a estrutura de repetição.
III. Um método é recursivo quando realizamos a chamada deste método dentro dele mesmo.

a) Apenas a alternativa I está correta.
b) Apenas a alternativa II está correta.
c) Apenas a alternativa III está correta.
d) Apenas as alternativas I e II estão corretas.
e) As alternativas I, II e III estão corretas.

6. Avalie as seguintes afirmações considerando os métodos recursivos:

I. Uma das regras é saber quando parar.
II. Uma das regras é saber decidir como fazer a próxima ação.
III. Uma das regras é quebrar uma jornada recursiva em um passo mais uma jornada recursiva menor.

Com base nas afirmações, é possível dizer que:

a) Apenas a alternativa I está correta.
b) Apenas a alternativa II está correta.
c) Apenas a alternativa III está correta.
d) Apenas as alternativas I e II estão corretas.
e) As alternativas I, II e III estão corretas.

Gabarito

1. **A**
   O módulo procedimento sem parâmetro é a estrutura que resolve um problema menor sem receber qualquer informação de outro lugar do algoritmo e não retorna valor para outro lugar do algoritmo.

2. **B**
   O módulo procedimento com parâmetro é a estrutura que resolve um problema menor recebendo alguma(s) informação(ões) de outro lugar do algoritmo e não retorna valor para outro lugar do algoritmo.

3. **C**
   O módulo função sem parâmetro é a estrutura que resolve um problema menor sem receber qualquer informação de outro lugar do algoritmo e retorna algum valor para outro lugar do algoritmo.

4. **D**
   O módulo função com parâmetro é a estrutura que resolve um problema menor recebendo alguma(s) informação(ões) de outro lugar do algoritmo e retorna algum valor para outro lugar do algoritmo.

5. **E**
   Um método é recursivo quando realizamos a chamada deste método, que pode ser função ou procedimento, dentro dele mesmo. Isto quer dizer que um método recursivo substitui uma estrutura de repetição.

6. **E**
   Quando desenvolvemos algoritmos recursivos, é importante se atentar para três regras importantes da recursão, são elas: 1) saber quando parar; 2) decidir como fazer a próxima ação; e 3) quebrar uma jornada recursiva em um passo mais uma jornada recursiva menor.

Para Pensar e Responder

Vamos exercitar a estrutura recursiva desenvolvendo um algoritmo que recebe do usuário dois números inteiros e calcula a somatória recursiva do primeiro número, o número de vezes do segundo número.

Por exemplo, se a entrada de dados for 2 e 10, você vai somar 2 + 2 + ... + 2, 10 vezes.

Desenvolva usando uma função e um procedimento e realize a chamada adequada de ambos os módulos.

Gabarito

```
// módulo função recursivo
numérico_inteiro SomaF (inteiro n1, inteiro n2)
  início
    Declarar
      s númerico_inteiro;
    se (n2 = 1)
      então
        retornar n1; // regra 1
      senão
        s <- n1 + SomaF(n1, n2-1); // regra 2 e 3
        retornar s;
    fimse;
  fim_módulo;


// módulo procedimento recursivo
SomaP (inteiro n1, inteiro n2, inteiro s)
  início
    Declarar

    se (n2 = 1)
      então
        escrever(“A soma com procedimento é “ , s+n1); // regra 1
      senão
        SomaP(n1, n2-1, n1+s); // regra 2 e 3
    fimse;
fim_módulo;

Algoritmo Somatoria
  início
    Declarar
      soma, nro1, nro2 numérico_inteiro;
    escrever("Digite dois inteiros");
    ler(nro1, nro2);
    soma <- SomaF(nro1, nro2);
    escrever("A soma com procedimento é " , soma);
    SomaP(nro1, nro2, 0);
  fim_algoritmo.
```