FIAP ON

ALGORITMOS:
APRENDA A PROGRAMAR

ESTRUTURAS DE REPETIÇÃO

JORGE L. SURIAN

4

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

LISTA DE FIGURAS

Figura 4.1 – Sintaxe da estrutura de repetição “para” 7
Figura 4.2 – Exemplo com dez repetições do comando imprima()............................7
Figura 4.3 – Exemplo de algoritmo de repetição que soma números entre dois números inteiros..9
Figura 4.4 – Exemplo de teste de mesa do algoritmo de repetição que soma números entre dois números inteiros 10
Figura 4.5 – Exemplo de algoritmo de repetição que soma quantos números inteiros o usuário quiser...11
Figura 4.6 – Exemplo de teste de mesa do algoritmo de repetição que soma quantos números inteiros o usuário quiser. 12
Figura 4.7 – Escola de Atenas (1509) affresco de Rafael, retratando provavelmente Euclides...21
Figura 4.8 – Exemplo que exibe números de 1 até 10 usando enquanto 23
Figura 4.9 – Exemplo que exibe números de 1 até 10 usando REPITA ... ATÉ........24
Figura 4.10 – Exemplo que exibe números de 1 até 10 usando FAÇA ... ENQUANTO 26
Figura 4.11 – Exemplo que exibe números de 1 até 10 usando LAÇO.....................28
Figura 4.12 – Descobrindo o maior divisor comum por força bruta...........................38
Figura 4.13 – Descobrindo o maior divisor comum usando o algoritmo de Euclides 39
Figura 4.14 – Descobrindo o maior divisor comum usando o algoritmo de Euclides e enquanto. 40
Figura 4.15 – Progressão aritmética com solução simples. 40
Figura 4.16 – Fatorial resolvido com algoritmo...42
Figura 4.17 – Fatorial resolvido com algoritmo transformado em função..................43
Figura 4.18 – Espiral resultante da sequência de Fibonacci 44
Figura 4.19 – Fibonacci resolvido com algoritmo 44
Figura 4.20 – Representação conceitual de um palíndromo 45
Figura 4.21 – Resolução de palíndromo. 47
Figura 4.22 – Algoritmo que determina se o número informado é primo ou não.......48
Figura 4.23 – Simulação do algoritmo de primos usando os valores 10 e 11. 49
Figura 4.24 – Algoritmo que determina se o número informado é primo ou não, melhorado. 50
Figura 4.25 – Nova simulação do algoritmo de primos usando os valores 10 e 11...51
Figura 4.26 – Algoritmo do número primo, melhorado novamente............................52
Figura 4.27 – Nova simulação do algoritmo de primos usando os valores 10 e 11...53
Figura 4.28 – Nova simulação do algoritmo de primos usando o valor 100 53
Figura 4.29 – Algoritmo do número primo, quarta versão 54
Figura 4.30 – Simulação do quarto algoritmo de primos usando vários valores 56
Figura 4.31 – Algoritmo do número primo, quinta versão..57
Figura 4.32 – Simulação do quinto algoritmo de primos usando vários valores........59
Figura 4.33 – Algoritmo do número primo, sexta versão..60
Figura 4.34 – Algoritmo do número primo, transformado em função. 82
Figura 4.35 – Algoritmo de cálculo fatorial, da forma convencional. 89
Figura 3.36 – Algoritmo de cálculo fatorial utilizando função recursiva 89
Figura 4.37 – Algoritmo de cálculo exponencial utilizando função recursiva.............91
Figura 4.38 – Torre de Hanói 92
Figura 4.39 – Representação gráfica e em linguagem natural para a Torre Hanoi...93

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Figura 4.40 – Exemplo de estratégia recursiva 93
Figura 4.41 – Mudança do disco maior diretamente da torre A para a torre C..........94
Figura 4.42 – Mudança dos dois discos da torre B para a torre C 94
Figura 4.43 – Torre de Hanói 95

FIAP

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

LISTA DE TABELAS

Tabela 4.1 – MDC entre dois números..22

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

SUMÁRIO

4 ESTRUTURAS DE REPETIÇÃO 6
4.1 Noções gerais de estruturas de repetição 6
4.2 Estrutura “Para” – sintaxe...6
4.2.1 Exemplo contagem de 1 a 10 7
4.3 Exemplos conceituais básicos...9
4.4 Exercícios básicos...12
4.5 Estruturas – Enquanto, Faça e Repita 20
4.5.1 Enquanto 22
4.5.1.1 Exemplo: contagem de 1 a 10..23
4.5.2 Sintaxe REPITA ... ATÉ 24
4.5.2.1 Exemplo: contagem de 1 a 10..24
4.5.3 Sintaxe FAÇA ... ENQUANTO 25
4.5.3.1 Exemplo: contagem de 1 a 10..26
4.6 Estruturas de repetição infinitas 27
4.6.1 Sintaxe de LAÇO 27
4.6.1.1 Exemplo: contagem de 1 a 10..28
4.7 Exercícios básicos...29
4.8 Exemplos de algoritmos com estruturas de repetição..............................37
4.9 Otimização de algoritmos 47
4.10 Exercícios de otimização..61
4.11 Exercícios gerais 63
4.12 Laços usando funções...81
4.12.1 Exercícios 82
4.13 Funções recursivas 87
4.13.1 Funções recursivas - definição 87
4.13.1.1 Definição 87
4.13.2 Exemplo: fatorial 88
4.13.3 Exemplo: potenciação 90
4.13.4 Exemplo: torre de Hanói 91
4.14 Exercícios clássicos 95
REFERÊNCIAS...104

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

4 ESTRUTURAS DE REPETIÇÃO

4.1 Noções gerais de estruturas de repetição

Consideremos o seguinte problema: “Obter a soma dos ‘n’ primeiros números pares positivos.” Se, por exemplo, n for igual a 6 então soma = 2+4+6+8+10+12, sendo o total 42.

Como dito no início deste capítulo, toda vez que tivermos uma **repetição**, não temos condições de solucionar o problema usando apenas os recursos conhecidos até o momento, especialmente em um problema tão dinâmico quando o sugerido: o número de repetições não é fixo, varia de acordo com o valor informado pelo usuário; um programador não teria condições de resolver o problema sem uma estrutura de repetição.

Existem, entretanto, várias estruturas ditas **de repetição** que permitem que um ou mais instruções sejam executadas mais de uma vez. Esse tipo de estrutura também é conhecido por outros nomes como **laços, loops, anéis**, entre outras maneiras de se referenciar a ela.

A questão que fica pendente é saber como simular uma somatória, isto é, a repetição de um processo de somas consecutivas de um conjunto de números. Vamos deixar pendente, por ora, esse problema.

4.2 Estrutura “Para” – sintaxe

Uma das estruturas de repetição mais importante é justamente a estrutura “Para”.

De um modo geral, ela “ordena” que **para um determinado número de situações**, um processo deverá ser repetido. Essa situação pode ser comparada a uma simples contagem. Imagine uma criança recebendo a ordem de seus pais para contar até 10 para um casal de amigos.

Os orgulhosos pais ficam completamente realizados quando a criança consegue cumprir a “tarefa”, não sem arrancar um bocejo dos infelizes amigos.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Mas como escrever isso numa linguagem algorítmica? Vamos à sua sintaxe:

Estrutura Para - Sintaxe

```
para <variável> de <índice de início > até <índice do final > passo <passo>
    <instrução >
fim-para
```

Figura 4.1 – Sintaxe da estrutura de repetição “para”
Fonte: Autor (2015)

4.2.1 Exemplo contagem de 1 A 10

Solução:

```
algoritmo conta_10_para;
variáveis
    i : inteiro;
fim-variáveis
início
    para i de 1 até 10 faça
        imprima(i);
    fim-para
fim
```

Figura 4.2 – Exemplo com dez repetições do comando imprima()
Fonte: Autor (2015)

No exemplo em questão temos uma variável “i” que faz a contagem. Por isso mesmo, chamamos a esse tipo de variável de variáveis de contagens ou, mais usualmente, de **contadores.**

A escolha da variável “i” não é arbitrária. Nós vamos perceber, ao longo do estudo, que as variáveis são tão importantes em nossos algoritmos, como também nos programas, que costumam ter letras ou palavras associadas as suas funções. Espera-se de uma variável “soma” que armazene uma soma, assim como das variáveis “a”, “b” e “c” que armazenem números arbitrariamente escolhidos pelo usuário.

Repare também que o exemplo de repetição possui um número de voltas conhecido: para exibir os números de 1 a 10, o laço dará exatas dez voltas, ou seja, a instrução imprima será repetida dez vezes.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Então, qual seria o motivo de uma variável contadora chamar-se “i”? A questão remonta as origens da TI e de uma das principais primeiras linguagens de computador: o FORTRAN. Nessa linguagem, as variáveis tinham tipos definidos pelas letras com as quais iniciavam seus nomes. Assim é que as variáveis entre “i” (da palavra inglesa “i”nteger) e “n” (da palavra inglesa “n”umber) seriam sempre representações de números inteiros, sendo as demais reservadas a expressões reais.

Naturalmente, a primeira variável inteira seria “i”, portanto passou a ser quase natural que contadores fossem “i”, “i1” etc. Também pareceu natural, quando mais de um contador fosse usado num mesmo programa que esses fossem representados por “j”, “k” etc.

Essa é a origem mais aceita das variáveis de as contagens serem exatamente as mesmas usadas no FORTRAN. Como os primeiros professores de linguagem invariavelmente ensinavam algoritmos e FORTRAN, passou a ser quase natural que os algoritmos usassem a notação do FORTRAN.

Mesmo depois de a linguagem FORTRAN ter se tornado obsoleta, um de seus cânones parece ter o condão da vida eterna, pois os contadores parecem carregar para sempre essa marca.

Observações:

- Devemos notar que “i”, nosso contador, é totalmente controlado pela estrutura “**para**” que o inicializa, controla seu incremento (ou decremento) e seu término. Ou seja, todos os seus aspectos são de controle da instrução e não do programador.
- Existem linguagens de programação que permitem ao programador altere o valor do contador durante a execução de uma contagem, este é o caso da linguagem C. Outras, como o Pascal, não permitem que os contadores sejam manipulados por qualquer outra instrução do programa, senão o “**para**” que “controla” o contador.
- Há linguagens, como o PL/SQL, que vão ainda além, pois sequer exigem que o contador seja declarado. Basta enunciá-lo como contador da instrução “**para**”. Isso basta para a linguagem se incumbir de todos os

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

demais aspectos relacionados à variável, que também só existe durante o tempo em que o “**para**” que a originou estiver em execução.

4.3 Exemplos conceituais básicos

Elaborar algoritmo que apresente a soma entre dois números inteiros informados pelo usuário:

Solução:

```
algoritmo soma_Para;
variáveis
   i,a,b,s : inteiro;
fim-variáveis
início
    a := leia();
    b := leia();
    s := 0;
    para i de a até b faça
        s := s + i;
    fim-para
      imprima(s);
fim
```

Figura 4.3 – Exemplo de algoritmo de repetição que soma números entre dois números inteiros.
Fonte: Autor (2015)

Há muitos aspectos importantes e fundamentais a serem observados nesse algoritmo, vamos a eles:

- Este algoritmo possui um número de voltas fixo, mas não conhecido; o número de volta é determinado pela entrada de dados do usuário, sendo assim, ele pode não dar volta alguma ou bilhões de voltas (o limite se torna o número máximo que pode ser armazenado em b).
- Observe que a variável “s” representa uma somatória, isto é, como toda soma, deve ser iniciada com zero. Esse tipo de variável é conhecido como **acumulador** ou variável para somatória ou ainda somatório.
- Perceba que a instrução “s := s + i” recebe a cada passagem seu próprio valor acrescido do valor de “i”. Ou seja, uma somatória só faz sentido se iniciada como zero, sempre agregar seu próprio valor à soma do número corrente a ser somado.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

- Finalmente, devemos observar que o valor da soma a ser impressa aparece apenas depois do *loop,* ou processo de repetição, ter-se esgotado.

Simulação

Suponha que o usuário digite os números inteiros 5 e 9, respectivamente. O algoritmo processará da seguinte maneira:

A	B	S	I	Impresso
5	9	0		
			5	
		5	6	
		11	7	
		18	8	
		26	9	
		35	10	35

Figura 4.4 – Exemplo de teste de mesa do algoritmo de repetição que soma números entre dois números inteiros
Fonte: Elaborado pelo autor (2015)

Observe que o valor de "i" é incrementado, ou seja, acrescido de 1 até o momento em que supera o limite máximo definido. Dizemos, habitualmente, que nesse instante o "contador estourou", ou seja, a condição do laço é "i até b", que é igual a 10 e, portanto, chegou ao seu limite. Repare que o contador sempre termina valendo um número maior que o definido como seu limite; i termina valendo dez, número que não cumpre a condição estabelecida.

Há ainda a possibilidade de que alguma instrução "quebre" o loop, ou seja, finalize-o de forma incondicional a contagem, antes que esta seja finalizada. Essa é uma situação especial, pois "desestrutura" um código que se prefere estruturado. Esse tema será tratado muito adiante neste curso.

Elaborar um algoritmo que apresente a soma entre "n" números inteiros informados pelo usuário.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Solução:

```
algoritmo soma_n;
variáveis
    i,n,c,s : inteiro;
fim-variáveis
início
    n := leia();
    s := 0;
      para i de 1 até n faça
        c := leia();
        s := s + c;
    fim-para
    imprima(s);
fim
```

Figura 4.5 – Exemplo de algoritmo de repetição que soma quantos números inteiros o usuário quiser.
Fonte: Autor (2015)

Novamente, temos muitos aspectos importantes e fundamentais a serem observados nesse algoritmo, vamos a eles:

- Observe que, da mesma forma como ocorreu no algoritmo anterior, a variável “s” representa uma somatória, isto é, deve ser iniciada com zero.
- Perceba que a instrução “s := s + c” recebe a cada passagem seu próprio valor acrescido do valor de “c”, que nada mais é que cada número “chutado” ou atribuído pelo usuário.
- Da mesma maneira que no exemplo anterior, o valor da soma a ser impressa aparece apenas depois do *loop,* ou processo de repetição, ter-se esgotado.

Simulação

Suponha que o usuário deseje informar cinco números, e informe isso ao algoritmo, e digite os seguintes números: 4, 5, 1, 8 e 3. O resultado seria:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

N	I	C	S	Impresso
5	1		0	
		4	4	
	2	5	9	
	3	1	10	
	4	8	18	
	5	3	21	
	6			21

Figura 4.6 – Exemplo de teste de mesa do algoritmo de repetição que soma quantos números inteiros o usuário quiser.
Fonte: Autor (2015)

Nessa simulação devemos compreender que, diferentemente da simulação anterior, agora uma variável “c” é alimentada arbitrariamente por um usuário e seu conteúdo é acumulado na variável “s”. Nesse exemplo, temos quatro tipos distintos de variáveis, a saber:

- “n”: Nosso número de elementos.
- “i”: Nosso contador, que conta de 1 até o número de elementos, no caso em questão “n”.
- “c”: Chute, candidato, semente. Trata-se de um valor atribuído arbitrariamente pelo usuário do programa ou quem quer que seja que esteja testando o algoritmo.

4.4 Exercícios básicos

a) Elabore um algoritmo que solicite ao usuário “n” números inteiros e imprima o maior deles em tela. Assuma que os números digitados são diferentes entre si.

```
algoritmo "Exercicio_Básicos_Capitulo_3_a"


var


   n: inteiro;

   i: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    numero: inteiro;
    maior: inteiro;

  inicio

    escreva("Informe o numero de elementos que serão digitados: ")
    leia(n)

    para i de 1 ATE n FACA
       escreva ("Informe o ", i, "° número inteiro positivo: ")
       leia (numero)
       se numero > maior ENTÃO
          maior := numero
       fimse
    fimpara
    escreva ("O maior número digitado foi ", maior)

  fimalgoritmo
```

b) Elabore um algoritmo que solicite ao usuário “n” números inteiros e imprima o maior deles em tela caso haja um que seja maior que outro (há uma possibilidade dos três serem iguais!)

```
algoritmo "Exercicio_Básicos_Capitulo_3_b"

var

   n: inteiro;
   i: inteiro;
   numero: inteiro;
   maior: inteiro;
   repetido: logico;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
inicio

  escreva("Informe o numero de elementos que serão digitados: ")
  leia(n)

  para i de 1 ATE n FACA
    escreva ("Informe o ", i, "° número inteiro positivo: ")
    leia (numero)
    se numero = maior ENTÃO
      repetido := verdadeiro
    fimse
    se numero > maior ENTÃO
      maior := numero
    fimse
  fimpara
  escreval ("O maior número digitado foi ", maior)
  se (repetido) ENTÃO
    escreva ("Existem números repetidos na sequencia de números")
  fimse

fimalgoritmo
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

c) Elabore um algoritmo que solicite ao usuário “n” números inteiros e imprima a soma de todos, exceto o menor deles. Assumir que os números digitados serão diferentes entre si.

```
algoritmo "Exercicio_Básicos_Capitulo_3_c"

var

   n: inteiro;

   i: inteiro;

   numero: inteiro;

   menor: inteiro;

   soma: inteiro;

inicio

   menor := 99999999

   escreva("Informe o numero de elementos que serão digitados: ")

   leia(n)

   para i de 1 ATE n FACA

      escreva ("Informe o ", i, "° número inteiro positivo: ")

      leia (numero)

      se numero < menor ENTÃO

         menor := numero

      fimse

      soma := soma + numero

   fimpara

   soma := soma - menor
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  escreval ("A soma dos numeros digitados sem considera o menor numero é ", soma)

fimalgoritmo
```

d) Elabore um algoritmo que solicite ao usuário “n” números inteiros, imprima a média desses números, exceto o menor deles. Assumir que os números digitados serão diferentes entre si.

```
algoritmo "Exercicio_Básicos_Capitulo_3_d"

var

   n: inteiro;

   i: inteiro;

   numero: inteiro;

   menor: inteiro;

   soma: inteiro;

   media: real;

inicio

   menor := 99999999

   escreva("Informe o numero de elementos que serão digitados: ")

   leia(n)

   para i de 1 ATE n FACA

      escreva ("Informe o ", i, "° número inteiro positivo: ")

      leia (numero)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
      se numero < menor ENTÃO
         menor := numero
      fimse
      soma := soma + numero
   fimpara
   soma := soma - menor
   media := soma / (n-1)
   escreval ("A media dos numeros digitados sem considera o menor numero é ", media)

fimalgoritmo
```

e) Elabore um algoritmo que solicite ao usuário dois números inteiros e imprima apenas os números pares existentes em seu intervalo (por exemplo, ao digitar 5 e 10, devem ser exibidos 6 e 8).

```
algoritmo "Exercicio_Básicos_Capitulo_3_e"

var

   i: inteiro;
   numero_1: inteiro;
   numero_2: inteiro;

inicio

   escreva("Informe um número inteiro: ")
   leia(numero_1)

   escreva("Informe um número inteiro maior que o número anterior: ")
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   leia(numero_2)


   escreval ("Listando os números pares entre ", numero_1 , " e ", numero_2 )


   para i de numero_1 + 1 ATE numero_2 -1 FACA

      se i MOD 2 = 0 ENTÃO

         escreval (i)

      fimse

   fimpara


fimalgoritmo
```

f) Elabore um algoritmo que solicite ao usuário dois números inteiros e imprima a soma dos pares existentes em seu intervalo (por exemplo, ao digitar 5 e 10, devem ser exibido 14).

```
algoritmo "Exercicio_Básicos_Capitulo_3_f"


var


   i: inteiro;

   numero_1: inteiro;

   numero_2: inteiro;

   soma: inteiro


inicio


   escreva("Informe um número inteiro: ")
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    leia(numero_1)

    escreva("Informe um número inteiro maior que o número anterior: ")
    leia(numero_2)

    para i de numero_1 + 1 ATE numero_2 -1 FACA
       se i MOD 2 = 0 ENTÃO
          soma := soma + i
       fimse
    fimpara

    escreva ("A soma dos pares é : ", soma)

fimalgoritmo
```

g) Elabore um algoritmo que solicite ao usuário dois números inteiros e imprima apenas aqueles em seu intervalo que terminarem em “7” (por exemplo, ao digitar 1 e 50, deverão ser exibidos os números 7, 17, 27, 37 e 47).

DICA PARA SOLUÇÃO: O último digito de qualquer número pode ser obtido através de uma simples conta: pegue o número e divida-o por 10, olhando para seu resto (MOD 10). Exemplos: 107 MOD 10 = 7 e 135112456 MOD 10 = 6.

```
algoritmo "Exercicio_Básicos_Capitulo_3_g"

var

   i: inteiro;
   numero_1: inteiro;
   numero_2: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
inicio

   escreva("Informe um número inteiro: ")
   leia(numero_1)

   escreva("Informe um número inteiro maior que o número anterior: ")
   leia(numero_2)

   escreval("Listando os números que terminam em 7")

   para i de numero_1 + 1 ATE numero_2 -1 FACA
      se i MOD 10 = 7 ENTÃO
         escreval (i)
      fimse
   fimpara

fimalgoritmo
```

4.5 Estruturas – Enquanto, Faça e Repita

Conforme vimos anteriormente, a estrutura "**PARA**" é muito adequada para situações em que sabemos quantas vezes um processo irá se repetir; o nome que damos a isso é **estrutura de repetição determinística**.

Mas e se encontramos problemas que, apesar de exigirem um processo de repetição para sua resolução, não tenham um número prévio de interações conhecidas?

Vamos nos valer de um dos algoritmos mais antigos conhecidos pela humanidade, para exemplificarmos essa situação: o algoritmo de Euclides.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Figura 4.7 – Escola de Atenas (1509) affresco de Rafael, retratando provavelmente Euclides
Fonte Wikimedia Commons (2010)

Motivação: Determinar o maior divisor comum entre dois números. Por exemplo, 10 é o maior divisor de 20 e 30 simultaneamente. Na verdade, 20 e 30 são divisíveis por 1 (aliás, quaisquer que sejam os números, na pior das hipóteses, 1 será divisor de ambos), 2, 5 e 10. Facilmente percebemos, nem que seja por testes de todas as possibilidades existentes, que 10 é o maior divisor comum entre 20 e 30.

Embora eficaz, dividir os dois números por todos os números menores que eles e posteriormente identificar o maior divisor comum entre os dois é um processo muito trabalhoso. Seres humanos gostam de atalhos, e os matemáticos não são exceções à regra.

Este atalho foi desenvolvido por um matemático grego chamado Euclides há 300 a.C., e à ele damos os nomes de “Algoritmo do MDC” ou “Algoritmo de Euclides”.

Procede-se da seguinte maneira: Divide-se o maior número pelo menor, mas o importante dessa divisão não é descobrir o divisor, mas o resto da divisão. Descarta-se o maior número, e divide-se novamente, desta vez usando o menor número e o resto da divisão anterior. Repete-se o processo até que o resto se torne zero.

A tabela abaixo nos mostra como encontrar o MDC de dois números:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	B A	B			
30	20	10	0		
	1	2			

Tabela 4.1 – MDC entre dois números
Fonte: FIAP (2015)

Note que "A" inicialmente vale 30 e "B" vale 20. O resto da divisão entre eles é 10. A variável "A" recebe o valor de "B" (20), descartando o 30. Em "B" guardamos o resto da divisão, 10. Ao se dividir novamente A por B, agora 20 dividindo o 10, o resto dá zero, logo, o maior divisor comum entre "30" e "20" é "10"! Ao se determinar o MDC de 80 e 54, dividimos um pelo outro, obtendo resto 26. Divide-se novamente, agora entre 54 e 26, obtendo-se resto 2; a divisão seguinte entre 26 e 2, obtemos um resto zero, concluindo que o MDC de 80 e 54 é 2.

A primeira coisa que observamos é não sabermos, de antemão, quantas vezes o processo irá se repetir. No primeiro exemplo precisamos realizar duas divisões, e no segundo exemplo foram três. Para outros números podem ser necessárias quatro ou cinco divisões e, para outros, a primeira basta. Isso nos impede o uso de uma estrutura "Para". Assim precisaremos de outro tipo de instruções, ou seja, as instruções **não determinísticas.**

4.5.1 Enquanto

A estrutura enquanto, por partir do pressuposto que uma condição terá que ser inicialmente válida para ser executada e também só continuará a ser executada enquanto permanecer verdadeira; é conhecida também como estrutura de repetição otimista**.**

Sintaxe:

```
enquanto  <condição> = verdade  faça
    <instrução>
fim-enquanto
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Vamos observar como montar uma contagem, semelhante como fizemos na estrutura **para,** somente que usando **enquanto**:

4.5.1.1 Exemplo: contagem de 1 a 10

Solução:

```
algoritmo conta_10_enquanto;
variáveis
    i : inteiro;
fim-variáveis
início
    i := 1;
    enquanto i <= 10 faça
        imprima(i);
        i := i + 1;
    fim-enquanto
fim
```

Figura 4.8 – Exemplo que exibe números de 1 até 10 usando enquanto
Fonte: Autor (2015)

Observações:

- Devemos notar que "i" continua sendo um mero contador, mas diferentemente do que ocorria nas estruturas "**para**", em "**enquanto**" o controle do início do contador e seu passo é encargo do programador e não da instrução.
- Tecnicamente, não estamos diante de uma contagem, mas de uma repetição que termina quando a variável "contadora" atingir determinado valor;
- A instrução "**enquanto**" precisa que seu contador seja inicialmente válido, para que seja executada numa primeira execução.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

4.5.2 Sintaxe REPITA ... ATÉ

A estrutura "**REPITA**" força a execução dos comandos que deve controlar ao menos por uma vez, pois faz o primeiro teste de situação **após** a última das instruções que estiverem alinhadas entre "Repita – Até".

Por partir do pressuposto que executará ações até uma condição ser satisfeita, é conhecida também como **estrutura de repetição pessimista**.

Sintaxe:

```
repita
      <instrução>
até <condição> = verdade;
```

Vamos observar como montar uma contagem, semelhante como fizemos nas estruturas **para** e **enquanto**.

4.5.2.1 Exemplo: contagem de 1 a 10

Solução:

```
algoritmo conta_10_repita;
variáveis
   i : inteiro;
fim-variáveis
início
  i := 1;
  repita
    imprima(i);
    i := i + 1;
  até i > 10;
fim
```

Figura 4.9 – Exemplo que exibe números de 1 até 10 usando REPITA ... ATÉ
Fonte: Autor (2015)

Observações:

- Devemos notar que "i" continua sendo um mero contador, mas comporta-se de maneira muito semelhante ao que ocorria na estrutura

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

"**ENQUANTO**", ou seja, o controle do início do contador e seu passo é encargo do programador e não da instrução.

- Tecnicamente, não estamos de uma contagem: a instrução será repetida até que a condição estabelecida se torne verdadeira, quando o laço é "quebrado". Entretanto, o programador deve ter em mente que se uma instrução "**REPITA**" nunca vier a se tornar verdadeira, o programa entrará num "loop infinito", ou seja, jamais irá parar, sem a intervenção de um usuário.
- A instrução "**REPITA**" não precisa que seu contador seja inicialmente válido, para que seja executada numa primeira execução. Isso acontece pois a verificação condicional acontece ao final do bloco de instruções, e não em seu início com nas estruturas de repetição apresentadas antes. Ou seja, não importa se a condição de repetição é válida ou não, temos a certeza absoluta que ao menos uma volta será dada;
- Nem todas as linguagens implementar a instrução "**REPITA**", Entre as linguagens de programação que usam "repita" podemos citar COBOL e Pascal. Entre as que não utilizam "repita", temos C e suas derivações (C#, PHP, Java etc.) e BASIC.

4.5.3 Sintaxe FAÇA ... ENQUANTO

Sintaxe:

```
faça
     <instrução>
enquanto <condição> = Verdade;
```

Da mesma maneira que "**REPITA",** a estrutura "**FAÇA**" garante que os comandos do bloco sejam executados ao menos uma vez, pois realiza o primeiro teste de situação **após** a última das instruções que estiverem alinhadas entre "**FAÇA... ENQUANTO**".

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Entretanto, assim como a instrução "**ENQUANTO**", permanece executando um comando enquanto uma condição permanecer verdadeira, "quebrando" seu laço no momento que a condição se torna falsa, ou seja, exatamente o oposto da instrução "**REPITA**".

Por partir do pressuposto que executará ações enquanto uma condição estiver satisfeita, é uma forma variante de "**ENQUANTO**", portanto também é **estrutura de repetição otimista**.

Vamos observar como montar uma contagem, semelhante como fizemos nas estruturas **para, enquanto e repita**.

4.5.3.1 Exemplo: contagem de 1 a 10

Solução:

```
algoritmo conta_10_faca;
variáveis
   i : inteiro;
fim-variáveis
início
  i := 1;
  faça
    imprima(i);
    i := i + 1;
  enquanto i <= 10;
fim
```

Figura 4.10 – Exemplo que exibe números de 1 até 10 usando FAÇA ... ENQUANTO
Fonte: Autor (2015)

Observações:

- Devemos notar que "i" continua sendo um mero contador, mas comporta-se de maneira muito semelhante ao que ocorria na estrutura "**ENQUANTO**", ou seja, o controle do início do contador e seu passo é encargo do programador e não da instrução.
- Tecnicamente, não estamos de uma contagem, mas de uma repetição que termina quando a variável "contadora" atingir determinado valor, como ocorria no enquanto, mas a instrução que faz esse controle está no **final** do loop e não em seu início, como ocorria no "**enquanto**".

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

- A instrução “**faça**” não precisa que seu contador seja inicialmente válido, para que seja executada numa primeira execução. O programador deve ter em mente que se uma instrução “**faça**” nunca vier a se tornar falsa, o programa entrará num “loop infinito”, ou seja, jamais irá parar, sem a intervenção de um usuário.
- Não estão disponíveis em todas as linguagens de programação, por exemplo, não aparecem em Pascal. Em linguagem C e suas derivadas como PHP ou Java, no entanto, são muito usadas.

4.6 Estruturas de repetição infinitas

É um tipo muito especial de estruturas de repetição, pois não preveem um término, portanto precisam ser “rompidas” pelo programador, quando da ocorrência de uma situação específica. É especialmente útil em situações nas quais é impossível ou pouco prático determinar a condição que levaria ao término de uma estrutura de repetição.

4.6.1 Sintaxe de LAÇO

Sintaxe

```
laço
     <instrução>
    se <condição> então
          Quebre
    fim-se
fim-laço
```

> **Atenção**: este é apenas um exemplo das diversas formas de empregar estruturas de repetição dentro da Lógica de Programação. Visto que cada aluno possui seu próprio raciocínio, este exemplo é apresentado para ajudá-lo a compreender melhor o fluxo e execução dos comandos e não se aplica ao nosso interpretador.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A estrutura “**laço**” força a execução dos comandos que deve controlar independente de qualquer condição, por isso faz-se obrigatória a presença de um teste, dentro do laço.

Esse tipo de estrutura não é determinística, otimista ou pessimista, apenas é **infinita**. Naturalmente, um *infinito* muito curioso, pois “dura” apenas até determinada condição causar seu rompimento.

Vamos observar como montar uma contagem, semelhante como fizemos nas estruturas **para, enquanto, repita e faça**.

4.6.1.1 Exemplo: contagem de 1 a 10

Solução:

```
algoritmo conta_10_laco;
variáveis
    i : inteiro;
fim-variáveis
início
     i :-1;
      laço
    imprima(i);
    i := i + 1;
      se i = 11 então
         interrompa;
      fim-se
    fim-laço
fim
```

Figura 4.11 – Exemplo que exibe números de 1 até 10 usando LAÇO
Fonte: Autor (2015)

Atenção: este é apenas um exemplo das diversas formas de empregar estruturas de repetição dentro da Lógica de Programação. Visto que cada aluno possui seu próprio raciocínio, este exemplo é apresentado para ajudá-lo a compreender melhor o fluxo e execução dos comandos e não se aplica ao nosso interpretador

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Observações:

- Devemos notar que “i” continua sendo um mero contador, mas comporta-se de maneira muito semelhante ao que ocorria na estrutura “enquanto”, “repita” ou “para”, mas todo o controle da estrutura passou a ser do programador.
- A instrução “**laço**” **não** precisa de qualquer condição para persistir; o que realmente precisa, no entanto, é de alguma instrução que force sua quebra ou parada, diferindo totalmente das demais instruções de repetição vistas até aqui.
- São encontradas em linguagens como o PL/SQL e facilmente são simuladas na Linguagem C e suas derivadas, assim como em Pascal e BASIC.

4.7 Exercícios básicos

Retome a lista de exercícios básicos apresentados anteriormente e os resolva novamente, trocando “para” por “enquanto”, “faça” e “repita”.

```
algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_a"

var

   n: inteiro;
   i: inteiro;
   numero: inteiro;
   maior: inteiro;

inicio

   escreva("Informe o numero de elementos que serão digitados: ")
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   leia(n)

   i := 1

   ENQUANTO i <= n FACA
      escreva ("Informe o ", i, "° número inteiro positivo: ")
      leia (numero)
      se numero > maior ENTÃO
         maior := numero
      fimse
      i := i + 1
   FIMENQUANTO

   escreva ("O maior número digitado foi ", maior)

fimalgoritmo

-----------------------------

algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_b"

var

   n: inteiro;
   i: inteiro;
   numero: inteiro;
   maior: inteiro;
   repetido: logico;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
inicio

   i := 1
   escreva("Informe o numero de elementos que serão digitados: ")
   leia(n)

   ENQUANTO i <= n FACA
      escreva ("Informe o ", i, "° número inteiro positivo: ")
      leia (numero)
      se numero = maior ENTÃO
         repetido := verdadeiro
      fimse
      se numero > maior ENTÃO
         maior := numero
      fimse
      i := i+1
   FIMENQUANTO

   escreval ("O maior número digitado foi ", maior)

   se (repetido) ENTÃO
      escreva ("Existem números repetidos na sequencia de números")
   fimse

fimalgoritmo

------------------
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_c"

var

   n: inteiro;
   i: inteiro;
   numero: inteiro;
   menor: inteiro;
   soma: inteiro;

inicio

   menor := 99999999
   i := 1

   escreva("Informe o numero de elementos que serão digitados: ")
   leia(n)

   ENQUANTO i <= n FACA
      escreva ("Informe o ", i, "° número inteiro positivo: ")
      leia (numero)
      se numero < menor ENTÃO
         menor := numero
      fimse
      soma := soma + numero
      i := i + 1
   FIMENQUANTO
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   soma := soma - menor

   escreval ("A soma dos numeros digitados sem considera o menor numero é ", soma)


fimalgoritmo


--------------------


algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_d"


var


   n: inteiro;

   i: inteiro;

   numero: inteiro;

   menor: inteiro;

   soma: inteiro;

   media: real;


inicio


   menor := 99999999

   i := 1


   escreva("Informe o numero de elementos que serão digitados: ")

   leia(n)


   ENQUANTO i <= n FACA
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
      escreva ("Informe o ", i, "° número inteiro positivo: ")

      leia (numero)

      se numero < menor ENTÃO

         menor := numero

      fimse

      soma := soma + numero

      i := i + 1

   FIMENQUANTO

   soma := soma - menor

   media := soma / (n-1)

   escreval ("A media dos numeros digitados sem considera o menor numero é ", media)


fimalgoritmo


--------------


algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_e"

var


   i: inteiro;

   numero_1: inteiro;

   numero_2: inteiro;


inicio


   escreva("Informe um número inteiro: ")

   leia(numero_1)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   i := numero_1

   escreva("Informe um número inteiro maior que o número anterior: ")
   leia(numero_2)

   escreval ("Listando os números pares entre ", numero_1 , " e ", numero_2 )

   ENQUANTO i <= numero_2 -1 FACA
      se i MOD 2 = 0 ENTÃO
         escreval (i)
      fimse
      i := i + 1
   fimenquanto

fimalgoritmo

-------------

algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_f"

var

   i: inteiro;
   numero_1: inteiro;
   numero_2: inteiro;
   soma: inteiro

inicio
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   escreva("Informe um número inteiro: ")

   leia(numero_1)

   i := numero_1 + 1


   escreva("Informe um número inteiro maior que o número anterior: ")

   leia(numero_2)


   ENQUANTO i <= numero_2 -1 FACA

      se i MOD 2 = 0 ENTÃO

         soma := soma + i

      fimse

      i := i + 1

   FIMENQUANTO


   escreva ("A soma dos pares é : ", soma)


fimalgoritmo


---------------


algoritmo "Exercicio_Básicos_Enquanto_Capitulo_3_g"


var


   i: inteiro;

   numero_1: inteiro;

   numero_2: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
inicio

   escreva("Informe um número inteiro: ")
   leia(numero_1)
   i := numero_1 + 1

   escreva("Informe um número inteiro maior que o número anterior: ")
   leia(numero_2)

   escreval("Listando os números que terminam em 7")

   ENQUANTO i <= numero_2 -1 FACA
      se i MOD 10 = 7 ENTÃO
         escreval (i)
      fimse
      i := i + 1
   FIMENQUANTO

fimalgoritmo
```

4.8 Exemplos de algoritmos com estruturas de repetição

Faremos agora alguns exemplos usando os recursos aprendidos, através de outros de mediana complexidade. As soluções deixam de ser naturais e passam a exigir uma maior capacidade de abstração. É uma prática bastante aceitável simular-se várias vezes para ajudar no entendimento.

Evite memorizar os exemplos (“decorar” a resolução). Lógica de programação exige criatividade para solução dos problemas. Concentre-se em entender bem

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

como cada estrutura funciona, e as soluções aprendidas o ajudarão na construção de novas estratégias em problemas futuros. A memorização, por outro lado, “engessará” seu raciocínio, atrapalhando ao invés de ajudar.

PROBLEMA A: Dados dois números naturais “m “e “n”, escreva um programa que calcule o máximo divisor comum deles;

SOLUÇÃO 1 DO PROBLEMA A: Tentativa e erro.

Nessa primeira abordagem, vamos desconsiderar completamente todo o esforço do matemático grego Euclides e identificar o maior divisor existente entre dois números usando a famosa estratégia da “força bruta”: dividir os dois números por TODOS os números entre o número 1 e o menor número entre os dois números informados; a última divisão da qual ambos não deixarem resto é, por definição, o maior divisor comum.

```
algoritmo mdc_Forca_Bruta;
variáveis
    i, a, b, menor, mdc : inteiro;
fim-variáveis
início
    a := leia();
    b := leia();
    se a > b então
        menor := a;
    senão
        menor := b;
    fim-se
      para i de 2 até menor faça
          se a % i = 0 e b % i = 0 então
              mdc := i;
          fim-se
    fim-para
   imprima("MDC: ", mdc);
fim
```

Figura 4.12 – Descobrindo o maior divisor comum por força bruta
Fonte: Elaborado pelo autor (2015)

Nessa estratégia, simplesmente vamos testando possibilidades até encontrarmos o número procurado. Vale notar que partimos do fato de que na pior

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

das hipóteses, o número 1 será o MDC, uma vez que qualquer número é divisível por 1.

Todavia, já observamos anteriormente que Euclides, muito tempo atrás, criou um algoritmo muito mais eficiente e que estudamos anteriormente, em seus aspectos numéricos, exclusivamente. Vamos agora implementar esse tradicional algoritmo.

SOLUÇÃO 2 DO PROBLEMA A: Solução usando algoritmo de Euclides

```
algoritmo mdc_euclides;
variáveis
    i, a, b, d, resto : inteiro;
fim-variáveis
início
    a := leia();
    b := leia();
      repita
         resto := a % b;
         a := b;
         b := resto;
      até resto = 0;
      imprima("MDC: ", a);
fim
```

Figura 4.13 – Descobrindo o maior divisor comum usando o algoritmo de Euclides
Fonte: Elaborado pelo autor (2015)

Nessa segunda estratégia, percebe-se que o número de iterações, isto é, o número de vezes que o algoritmo repete um bloco de instruções é muito menor do que a solução inicial, baseada em “força bruta”. Isso não é exatamente uma surpresa: métodos mais “diretos” ou um menor refinamento lógico costumam resultar em algoritmos fáceis de serem construídos, contudo pouco performáticos. A recíproca é também verdadeira, como o segundo algoritmo acaba de nos mostrar.

Além disso, a estratégia estabelecida por Euclides tem um número de divisões que varia dependendo dos números informados e, por isso deixamos de saber quantas vezes o processo se repetirá. Assim sendo, passamos a ser obrigados a usar uma estrutura de repetição não determinística como é o caso de “repita”, por exemplo.

Vamos retomar o problema e resolvê-lo com a instrução “enquanto”.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo mdc_euclides_enquanto;
variáveis
    i, a, b, d, resto : inteiro;
fim-variáveis
início
    a := leia();
    b := leia();
      resto := a % b;
      enquanto resto <> 0 faça
        a := b;
        b := resto;
        resto := a % b;
    fim-enquanto
      imprima("MDC: ", b);
fim
```

Figura 4.14 – Descobrindo o maior divisor comum usando o algoritmo de Euclides e enquanto.
Fonte: Elaborado pelo autor (2015)

PROBLEMA B: A série 4,7,10,13,16,19... é uma PA (Progressão Aritmética). Elabore um algoritmo que imprima os “n” primeiros termos dessa interessante série, que tem números primos, palíndromos, de Fibonacci, quadrados entre vários outros.

Solução

```
algoritmo pa_4_7_10;
variáveis
    n,a,b,c,i : inteiro;
fim-variáveis
início
    n := leia();
    a := 4;
    para i de i até n faça
         imprima(a);
         a := a + 3;
    fim-para
fim
```

Figura 4.15 – Progressão aritmética com solução simples.
Fonte: Elaborado pelo autor (2015)

Nessa solução, podemos observar que a variável “a” é iniciada com 4 e é incrementada de 3 em 3, reproduzindo com exatidão a proposta do exercício. Embora o número de termos seja variável e de acordo com o desejo do usuário, uma vez informado por ele, sabemos dessa vez quantos termos teremos que

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

processar, por isso usamos a instrução "para" indo até "n" números para solução desse problema.

Naturalmente, é possível solucionar essa questão com "repita", "faça", "enquanto" ou "laço" (vale a pena exercitar), mas a estrutura mais adequada para a necessidade é justamente a instrução "para", pois na maioria das linguagens seu processo de iteração é mais eficiente nessa instrução do que nas suas correlatas.

PROBLEMA C: Elabore um algoritmo que calcule o fatorial de um número inteiro qualquer.

RELEMBRAR é VIVER!

Espere... você lembra como se calcula um fatorial?

Vamos voltar no tempo, na época do seu ensino fundamental, quando o professor de matemática lhe pedia para calcular isso:

5!

Calcular o fatorial de 5 significa multiplicá-lo por todos os números menor que ele, até o número 1. Sendo assim:

5! = 5 x 4 x 3 x 2 x 1

5! = 20 x 3 x 2 x 1

5! = 60 x 2 x 1

5! = 120 x 1

5! = 120

Lembrou?

Solução

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo fatorial_simples;
variáveis
    n, p, i : inteiro;
fim-variáveis
início
    n := leia();
       p := 1;
       para i de 2 até n faça
        p := p * i;
           imprima(p);
    fim-para
fim
```

Figura 4.16 – Fatorial resolvido com algoritmo
Fonte: Elaborado pelo autor (2015)

Na solução apresentada, por um lado, como sabemos quantos termos terão que ser multiplicados, voltamos a usar a instrução "para". Por outro lado, destaca-se nesse algoritmo o fato da variável "p", aquela que irá armazenar o resultado do fatorial, ou seja, a produtória, por isso "p", é inicializada com 1, justamente por 1 (número um), ser o elemento neutro quando fazemos produtos (inicializa-la em 0 arruinaria todas as multiplicações posteriores, afinal, qualquer número multiplicado por zero é igual a zero. Além disso, deixar de inicializa-la antes do laço resultaria em erro na maioria das linguagens de programação).

A seguir, vamos transformar o algoritmo fatorial numa função que possa ser chamada por um programa principal.

Solução

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo fatorial_funcao;
variáveis
fim-variáveis
início
    imprima(fatorial(10));
fim
função fatorial (n: inteiro) : real
    i : inteiro;
    result: real;
início
    result := 1
    para i de 2 até n faça
        result := result * i;
          fatorial := result;
    fim-para
fim
```

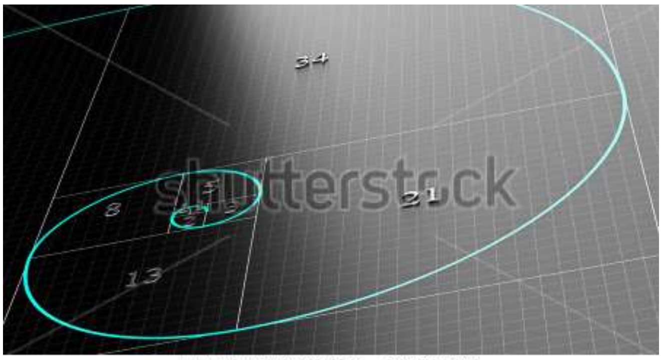
Figura 4.17 – Fatorial resolvido com algoritmo transformado em função
Fonte: Elaborado pelo autor (2015)

Transformar a solução em função torna o algoritmo extremamente reutilizar: Caso queiramos imprimir outros resultados de fatorial além de 10, como 5, por exemplo, basta acrescentarmos a instrução “`imprimir(fatorial(5));`” ao programa principal.

PROBLEMA D: Elabore algoritmo que apresente os primeiros “n” termos de Fibonacci

Os números de Fibonacci também são conhecidos como números da natureza, pois desde uma mera estrela do mar até a Via Láctea (constelação onde se encontra uma minúscula estrela que chamamos de Sol) seguem a “razão áurea” que se esconde atrás dos números dessa série, que os gregos usavam, por exemplo, em sua rica arquitetura. O matemático italiano Leonardo de Pisa o estudou, gerando a série que leva o nome do seu apelido (Fibonacci, filho de Bonaccio). A sequência começa por 0 e 1, na qual, cada termo subsequente corresponde a soma dos dois anteriores.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com



www.shutterstock.com · 385744144

Figura 4.18 – Espiral resultante da sequência de Fibonacci
Fonte: Banco de imagens Shutterstock (2016)

Uma das várias estratégias para obtenção de termos de Fibonacci consiste em, a partir dos números formadores da série, 0 e 1, ter o próximo termo da série, simplesmente somando-se esses dois números.

Ora, teremos então o número 1, novamente, como o terceiro número dessa série. O quarto número (o próximo, como será chamado doravante) será obtido pela soma do último (no caso 1) e do penúltimo (1, novamente), resultando em 2.

Esse processo será repetido sucessivamente, gerando:
0,1,1,2,3,5,8,13,21,34,55...

Solução

```
algoritmo fibonacci;
variáveis
    i, n, a, b, c : inteiro;
fim-variáveis
início
    imprima("Digite um Número: ");
    n := leia();
    a := 0;
    b := 1;
    imprima(a,b);
      para i de 1 até n faça
      c := a + b;
      imprima(c);
      a := b
      b := c;
    fim-para
fim
```

Figura 4.19 – Fibonacci resolvido com algoritmo
Fonte: Elaborado pelo autor (2015)

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Curiosamente, o número 13, considerado por muitos o “número do azar” faz parte da série. Podemos dizer que se trata de um número pelo qual a natureza teria uma certa preferência, ao contrário de 7 (habitualmente conhecido como número da “sorte”) ou 4 (cujo trevo, traria “sorte”).

Devemos observar que um pensamento linear nos levaria a criar múltiplas variáveis, sempre se somando as anteriores, algo como “c := a + b; d := b + c; e := c + d”, e assim sucessivamente, numa repetição sem fim e totalmente impraticável, se pensarmos na escrita de um algoritmo para solução dessa questão.

Para se obter nosso quarto termo precisamos apenas somar o segundo e o terceiro termos, assim sendo, podemos “descartar” o primeiro termo (nosso “a”), substituindo pelo segundo termo (nosso “b”). A variável “b” continua com o segundo número, descartamos o segundo termo que está em “b”, ficando em seu lugar o terceiro termo. Ora, “c” passa a ser o quarto termo, pois recebe a soma de “a” e “b” a cada iteração, resolvendo o problema.

PROBLEMA E: Elabore algoritmo que decida se um número informado pelo usuário é palíndromo

Palíndromos são números que lidos da esquerda para a direita, como normalmente lemos os números, ou de forma invertida, sempre apresentam o mesmo valor.

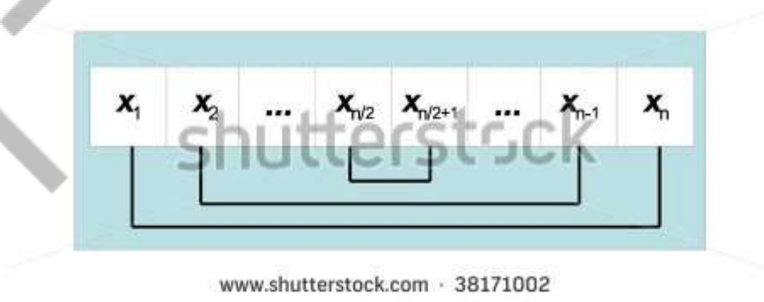


Figura 4.20 – Representação conceitual de um palíndromo
Fonte: Banco de imagens Shutterstock (2016)

FASCINANTES TRUQUES MATEMÁTICOS

Somos capazes de “pegar” qualquer dígito em um número de vários dígitos, apenas com algumas divisões estratégicas utilizando como divisores o

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

número 10 e seus múltiplos.

Exemplo:

15 ... como extrair o “1” e o “5”?

Para “pegar” um 1, basta dividir o número por 10 desprezando o resto:

15 DIV 10 = 1

Para “pegar” o 5, divide-se também por 10 mas, desta vez, só considero o resto:

15 % 10 (ou 15 MOD 10) = 5.

E no caso de um número maior, como 500109?

Se quisermos extrair o “5” (primeiro digito), devemos dividi-lo por 100000 (é um número de 6 dígitos e quero o primeiro, logo dividimos por 1 seguido de “6-1” zeros:

500109 DIV 100000 = 5

E para extrair o 9? Dividimos novamente por 10, olhando apenas para o resto:

500109 % 10 = 9

Para extrair o 1, precisamos de duas divisões: transformarmos o número em 5001 para depois capturar o último dígito:

500109 DIV 100 = 5001
5001 % 10 = 1

A estratégia pode ser invertida: transformar o número em 109 e, depois, pegamos o número 1:

500109 % 1000 = 109
109 DIV 100 = 1

Incrível, não?

Solução

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo palindromo;
variáveis
    valor, num, inv, resto : inteiro;
fim-variáveis
início
   inv := 0;
   imprima("Digite um numero: ");
   valor := leia();
   num := valor;
   enquanto (num > 0) faça
        resto := num % 10;
        num := num div 10
        inv := inv * 10 + resto;
           se valor = inv então
             imprima(valor, "é Palindromo");
           senão
              imprima(valor, " é comum");
        fim-se
    fim-enquanto
fim
```

Figura 4.21 – Resolução de palíndromo.
Fonte: Elaborado pelo autor (2015)

Nossa estratégia, dessa vez, foi replicar o valor da variável que recebe o número, pretensamente palíndromo. A partir da variável em que esse número está copiado, extraímos dígito a dígito, multiplicando o número resultante por dez, de forma a “inverter” o número em questão.

4.9 Otimização de algoritmos

A partir de agora, passamos a ser capazes de construir algoritmos com um certo grau de complexidade, mas não podemos evoluir neste aspecto sem nos preocupamos com a qualidade do código.

Abordamos levemente o assunto quando estudamos o problema do maior entre três números distintos, todavia, o foco da discussão naquela ocasião era muito mais conhecer formas distintas de pensamento do que efetivamente pensar em qual algoritmo era superior.

Voltemos agora ao problema da otimização de algoritmos com muito mais profundidade.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

PROBLEMA A: Elabore um algoritmo que, ao usuário informar um número inteiro qualquer, indique se esse número é primo ou não.

SOLUÇÃO 1 DO PROBLEMA A

Todo número “n” que não seja divisível por nenhum número entre 2 e “n-1” é primo, sendo assim, primos são divisíveis apenas um 1 e si mesmos.

Sendo assim, nessa primeira estratégia, testaremos todos os números num intervalo previamente conhecido, tomando uma decisão bastante simples no final: encontrado um divisor de “n”, saberemos que “n” não é primo. E vice-versa... ou seja, o método da “força bruta”.

O algoritmo a seguir resolve a questão da forma como estamos propondo.

```
algoritmo primo_1;
variáveis
    a, i : inteiro;
    primo : lógico;
fim-variáveis
início
    imprima("Digite um Número: ");
    a := leia();
    primo := verdadeiro;
      para i de 2 até a - 1 faça
        se a % i = 0 então
               primo := falso;
        fim-se
    fim-para
    se primo = verdadeiro então
           imprima(a," é primo!");
    senão
           imprima(a," não é primo!");
    fim-se
fim
```

Figura 4.22 – Algoritmo que determina se o número informado é primo ou não.
Fonte: Elaborado pelo autor (2015)

Vamos agora simular para 10 (não primo) e 11 (primo), visando entender o funcionamento do algoritmo.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
11		Verdadeiro	
	2		
	3		
	4		
	5		
	6		
	7		
	8		
	9		
	10		
			11 é primo

A	I	Primo	Impresso
10		Verdadeiro	
	2	Falso	
	3		
	4		
	5	Falso	
	6		
	7		
	8		
	9		10 não é primo

Figura 4.23 – Simulação do algoritmo de primos usando os valores 10 e 11.
Fonte: Elaborado pelo autor (2015)

Quando analisamos as simulações, notamos que, no caso em que o número é primo, ou seja, 11, foram testados números até 10. Todavia, é muito fácil perceber que não encontraremos nenhum divisor de um número que supere sua metade. Não precisamos testar todos até “n-1”, apenas até sua metade (“n/2”):

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo primo_2;
variáveis
    a, i : inteiro;
    primo : lógico;
fim-variáveis
início
    imprima("Digite um número: ");
      a := leia();
      primo := verdadeiro;
      para i de 2 até a/2 faça
        se a % i = 0 então
               primo := falso;
        fim-se
    fim-para
    se primo = verdadeiro então
           imprima(a," é primo!");
    senão
           imprima(a," não é primo!");
    fim-se
fim
```

Figura 4.24 – Algoritmo que determina se o número informado é primo ou não, melhorado.
Fonte: Elaborado pelo autor (2015)

Uma pequena alteração, porém, com grandes resultados. Voltemos à nossa simulação:

A	I	Primo	Impresso
			Digite um numero:
11		Verdadeiro	
	2		
	3		
	4		
	5		11 é primo

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
			Digite um numero:
10		Verdadeiro	
	2	Falso	
	3		
	4		
	5	Falso	10 não é primo

Figura 4.25 – Nova simulação do algoritmo de primos usando os valores 10 e 11.
Fonte: Elaborado pelo autor (2015)

É fácil perceber que o esforço foi reduzido à metade, nos dois casos! Começamos a perceber que, embora existam várias soluções para um mesmo problema, seu desempenho pode ser muito, mas muito diferente!

Entretanto, antes de nos contentarmos com a nova solução, melhor seria observar que continuamos a desperdiçar comparações. Vamos pensar no número 10. Ora, já no primeiro teste é possível identificá-lo como não primo. E não apenas o número 10... Todos os números pares! Ou seja, para identificarmos que 1000 não é primo em vez de 499 testes, um apenas bastaria...

Chegamos então diante de uma importante encruzilhada. Se mantivermos a estrutura “para” faremos inúmeros testes inúteis, que poderiam ser facilmente evitados se simplesmente substituíssemos a instrução “para” por outra da família dos loops não determinísticos.

Tentemos então uma solução usando a instrução “enquanto”...

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo primo_3;
variáveis
    a, i : inteiro;
    primo : lógico;
fim-variáveis
início
    imprima("Digite um Número: ");
      a := leia();
      primo := verdadeiro;
      i := 2;
      enquanto (i <= a / 2) e (primo = verdadeiro) faça
        se a % i = 0 então
               primo := falso;
        fim-se
        i := i + 1;
    fim-enquanto
      se primo = verdadeiro então
         imprima(a," é primo!");
      senão
         imprima(a," não é primo!");
    fim-se
fim
```

Figura 4.26 – Algoritmo do número primo, melhorado novamente.
Fonte: Elaborado pelo autor (2015)

Vamos analisar, atentamente, o que aconteceu com nossa simulação.

A	I	Primo	Impresso
			Digite um número:
11		Verdadeiro	
	2		
	3		
	4		
	5		
	6		11 é Primo

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
			Digite um número:
10		Verdadeiro	
	2	Falso	
			10 não é Primo

Figura 4.27 – Nova simulação do algoritmo de primos usando os valores 10 e 11
Fonte: Elaborado pelo autor (2015)

A	I	Primo	Impresso
			Digite um número:
100		Verdadeiro	
	2		
		Falso	
			100 não é primo

Figura 4.28 – Nova simulação do algoritmo de primos usando o valor 100
Fonte: Elaborado pelo autor (2015)

Tivemos um incrível aumento de performance. Tanto é que “ousamos” testar um número razoavelmente grande, o 100. Facilmente, ou seja, no primeiro teste, percebemos que 100 não é primo. Todavia, se tentássemos analisar o número 101, perceberíamos facilmente que ainda necessitaríamos de 49 testes. Nossa simulação, sem dúvida, ficaria bastante extensa. Impraticável, se pensarmos em reproduzi-la “na mão”.

Mas, estudando um pouco mais de cálculo, descobriremos que não é preciso testar “candidatos a divisores de um número até sua metade”. Basta testarmos até o número inteiro menor que sua própria raiz quadrada!

Ora, a questão do 101 se reduz drasticamente, pois em vez de 49 testes, faríamos apenas 9 testes, isto é, testaríamos números entre 2 e 10 somente.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo primo_4;
variáveis
    a, i : inteiro;
    primo : lógico;
fim-variáveis
início
    imprima("Digite um Número: ");
    a := leia();
    primo := verdadeiro;
    i := 2;
    enquanto (i <= raizq(a)) e (primo = verdadeiro) faça
        se a % i = 0 então
               primo := falso;
        fim-se
        i := i + 1;
    fim-enquanto
      se primo então
        imprima(a," é primo!");
      senão
        imprima(a," não é primo!");
    fim-se
fim
função raizq(numero : inteiro) : inteiro
    raiz : inteiro;
início
     raiz := 0;
    repita
        raiz := raiz + 1;
    até(raiz * raiz) > numero;
        raiz := raiz - 1;
    retorne raiz;
fim
```

Figura 4.29 – Algoritmo do número primo, quarta versão
Fonte: Elaborado pelo autor (2015)

Vamos nos deter, exclusivamente na situação em que temos números primos a serem testados. Salta aos olhos que a redução para identificarmos que 101 é primo, comparando o quarto algoritmo com o primeiro. Muitas vezes, no ambiente profissional encontramos a mesma situação, pois quase sempre usuários de sistemas sabem regras “mais objetivas” do que alguma regra conhecida pelo pessoal de TI. Nesse caso, um especialista em matemática, certamente sabe que não precisamos testar um número até sua metade. Todavia, o que é obrigação para um especialista em matemática, não o é, para um profissional de TI.

Vejamos como fica nossa simulação:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
			Digite um número:
11		Verdadeiro	
	2		
	3		
	4		
	5		
	6		
	7		
	8		
	9		
	10		
			101 é Primo

A	I	Primo	Impresso
			Digite um número:
11		Verdadeiro	
	2		
	3		
	4		11 é primo

A	I	Primo	Impresso
			Digite um número:
10		Verdadeiro	
	2		
	3	Falso	
			10 não é primo

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
			Digite um número:
100		Verdadeiro	
	2		
	2	Falso	
	3	Falso	
			100 não é primo

Figura 4.30 – Simulação do quarto algoritmo de primos usando vários valores
Fonte: Elaborado pelo autor (2015)

Embora os ganhos sejam evidentes, será que não seria possível melhorarmos ainda mais nosso algoritmo?

Vamos começar observando que se o número 2 liquida uma série de “candidatos” a primo (metade dos números inteiros, rigorosamente) e o 3 termina com a pretensão de vários outros “candidatos”, o 4, na prática, não elimina nenhum candidato.

Isso ocorre porque o número 4 e todos os demais números pares, simplesmente “eliminam” números pares, mas não há qualquer número par a ser eliminado, pois o 2 já os eliminou. O mesmo ocorre com o 6, com o 8, com o 10, ou seja, metade dos “pretensos” candidatos que temos, são testes inúteis, na prática.

E como resolver essa questão?

Ora, parece uma boa ideia eliminarmos dos testes todos os números pares, exceto o número 2. Chegaríamos, portanto no algoritmo que segue:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo primo_5;
variáveis
  a, i : inteiro;
  primo : lógico;
fim-variáveis
início
      imprima("Digite um Número: ");
      a := leia();
      primo := verdadeiro;
      i := 2;
     enquanto (i <= raizq(a)) e ( primo = verdadeiro) faça
        se a % i = 0 então
              primo := falso;
        fim-se
            se i % 2 = 0 então
               i := i + 1;
        senão
               i := i + 2;
        fim-se
    fim-enquanto
      se primo então
        imprima(a," é primo!");
      senão
        imprima(a," não é primo!"');
    fim-se
fim
função raizq(numero : inteiro) : inteiro
    raiz : inteiro;
início
     raiz := 0;
    repita
        raiz := raiz + 1;
    até(raiz * raiz) > numero;
        raiz := raiz - 1;
    retorne raiz;
fim
```

Figura 4.31 – Algoritmo do número primo, quinta versão
Fonte: Elaborado pelo autor (2015)

Vejamos como fica nossa simulação:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
			Digite um número:
11		Verdadeiro	
	2		
	3		11 é primo

A	I	Primo	Impresso
			Digite um número:
10		Verdadeiro	
	2	Falso	
	2	Falso	
	3	Falso	
	3	Falso	

A	I	Primo	Impresso
			Digite um número:
100		Verdadeiro	
	2		
	3	Falso	
			100 não é primo

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A	I	Primo	Impresso
			Digite um número:
11		Verdadeiro	
	2		
	3		
	4		
	5		
	6		
	7		
	8		
	9		
	10		
			101 é Primo

Figura 4.32 – Simulação do quinto algoritmo de primos usando vários valores
Fonte: Elaborado pelo autor (2015)

De fato, notamos nova melhoria, bastante significativa, nos casos em que o número analisado é primo e relativamente grande. Embora a simulação não venha a ser alterada, podemos melhorar a performance desse algoritmo, ainda que levemente. Vejamos como:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo primo_6;
variáveis
  a, i : inteiro;
  primo : lógico;
fim-variáveis
início
    imprima("Digite um Número: ");
    a := leia();
    primo := verdadeiro;
    i := 3;
    se a % 2 = 0 então
        primo := falso;
    fim-se
    enquanto (i <= raizq(a)) e (primo = verdadeiro) faça
        i := i + 2;
        se a % i = 0 então
               primo := falso;
        fim-se
    fim-enquanto
        se primo então
            imprima(a," é primo!");
        senão
            imprima(a," não é primo!");
        fim-se
fim
função raizq(numero : inteiro) : inteiro
    raiz : inteiro;
início
     raiz := 0;
    repita
        raiz := raiz + 1;
    até(raiz * raiz) > numero;
        raiz := raiz - 1;
    retorne raiz;
fim
```

Figura 4.33 – Algoritmo do número primo, sexta versão
Fonte: Elaborado pelo autor (2015)

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Embora pouco significativa, a economia de uma comparação dentro de um laço é sempre digna de nota. Na prática, os ganhos em economia de processamento são quase insignificantes, mas trata-se de uma boa prática de programação a ser sempre trabalhada.

Mas, poderíamos chegar a um sétimo aperfeiçoamento? A resposta, embora seja sim, nos leva a refletir sobre um ditado popular, muito recorrente em áreas como a engenharia: “O ótimo é inimigo do bom”.

Pensemos um pouco...

Ora, se evitamos testar candidatos a primos dividindo-os por múltiplos de 2, pois claramente qualquer número que seja divisível por 4, 6, 8 etc., também o será por 2. Mas, o que dizer do 9? Embora não seja múltiplo de 2, é de 3. Vale a mesma regra, ou seja, múltiplos de 3, como 9, 27 etc., também não são reais candidatos a divisores, isto é, os números divisíveis por esses números (9, 27), também são divisíveis por 3. Analogamente, eliminamos o 25 (em função do 5), o 49 (em função do 7) e fica fácil concluir que **somente números primos** são necessários para identificar se um número qualquer **é primo!**

Ora, então bastaria termos os números primos previamente armazenados na memória, por exemplo, e somente testarmos por esses números previamente armazenados. Embora esse algoritmo seja um bom teste de lógica, estamos diante de um caso em que uma pretensa melhoria, acaba na prática piorando (e muito, diga-se) o desempenho do algoritmo.

Isso ocorre porque será necessário muito esforço para previamente calcular e armazenar esse conjunto de números primos a serem testados. Com certeza, vale bem mais a pena realizar alguns testes inúteis (9, 15, 21, 25 etc.) do que armazenar números primos em memória **antes** de efetivamente iniciarmos nossos testes.

4.10 Exercícios de otimização

Retome os exercícios que você já fez e pense em uma forma de melhorar ao menos alguns deles.

```
algoritmo "Exercicio_otimizado"
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
var

   n, i, numero, maior: inteiro;

inicio

   escreva("Informe o numero de elementos que serão digitados: ")
   leia(n)

   i := 1

   ENQUANTO i <= n FACA
      escreva ("Informe o ", i, "° número inteiro positivo: ")
      leia (numero)
      se numero > maior ENTÃO
         maior := numero
      fimse
      i := i + 1
   FIMENQUANTO

   escreva ("O maior número digitado foi ", maior)

fimalgoritmo
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

4.11 Exercícios gerais

a) Elabore um algoritmo que calcule e exiba a soma dos “n” primeiros números inteiros positivos, sendo “n” informado pelo usuário. Exemplo, se n = 4, deverá ser impresso 10 (4+3+2+1).

```
algoritmo "Exercicio_Gerais_Capítulo_3_a"

var

   i: inteiro;
   soma: inteiro;
   numero: inteiro;

inicio

   escreva("Informe um número inteiro: ")
   leia(numero)

   escreval("Listando a soma dos números inteiros")

   PARA i de 1 ATE numero FACA
      soma := soma + i
   FIMPARA

   escreva (soma)

fimalgoritmo
```

b) Elabore um algoritmo que receba do usuário um número natural “n”, calcule a exiba o fatorial deste número.

```
algoritmo "Exercicio_Gerais_Capítulo_3_b"

var

   i: inteiro;
   fat: inteiro;
   numero: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade n.ferreira.tavares@avanade.com

```
inicio

   fat := 1

   escreva("Informe um número inteiro: ")
   leia(numero)

   escreval("Exibindo o fatorial")


   PARA i de 1 ATE numero FACA
      fat := fat * i
   FIMPARA

   escreva (fat)

fimalgoritmo
```

c) Um número é triangular se ele é produto de três números naturais consecutivos. Exemplo: 6 é triangular, pois 1 * 2 * 3 = 6. Elabore um algoritmo que receba do usuário um número inteiro “n”, por hipótese maior que zero, calcule e imprima os “n” primeiros números triangulares.

```
algoritmo "Exercicio_Gerais_Capítulo_3_c"

var

   i: inteiro;
   tri: inteiro;
   numero: inteiro;

inicio

   escreva("Informe um número inteiro: ")
   leia(numero)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    escreval("Exibindo os numeros triangulares")

    PARA i de 1 ATE numero-2 FACA
       tri := i
       tri := tri * (i + 1)
       tri := tri * (i + 2)
       escreval (i, " * ", i+1, " * ",i+2, " = ",tri)
    FIMPARA


fimalgoritmo
```

d) Elabore um algoritmo que receba do usuário um número inteiro “n” e devolva se “n” é triangular ou não.

```
algoritmo "Exercicio_Gerais_Capítulo_3_d"

var

    i: inteiro;
    numero: inteiro;

inicio

    escreva("Informe um número inteiro: ")
    leia(numero)

    PARA i de 1 ATE numero FACA
       se ( i*(i+1)*(i+2) = numero ) ENTÃO
          escreval ("O numero é triangular")
       fimse
    FIMPARA


fimalgoritmo
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade n.ferreira.tavares@avanade.com

e) Elabore um algoritmo que receba do usuário um número inteiro “n” informe se este é ou não primo.

```
algoritmo "Exercicio_Gerais_Capítulo_3_e"

var

   i: inteiro;
   numero: inteiro;
   contador: inteiro;

inicio

   escreva("Informe um número inteiro: ")
   leia(numero)

   PARA i de 1 ATE numero FACA
      SE numero MOD i = 0 ENTÃO
         contador := contador + 1
      FIMSE
   FIMPARA

   SE contador = 2 ENTÃO
      escreva ("O Número é primo")
   FIMSE

fimalgoritmo
```

f) Elabore um algoritmo que receba do usuário “n” números inteiros (quanto ele quiser), calcule e exiba a somatória desses números (Exemplo, o usuário decide informar 3 números, informando 5, 6 e 2, o algoritmo deve exibir 13).

algoritmo "Exercicio_Gerais_Capítulo_3_f"

var

i: inteiro;

numero: inteiro;

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  valor: inteiro;
  soma: inteiro;

inicio

  escreva("Informe a quantidade de números que deseja somar: ")
  leia(numero)

  PARA i de 1 ATE numero FACA
    escreva ("Informe um número para a soma ")
    leia (valor)
    soma := soma + valor
  FIMPARA

  escreva ("A soma é : ", soma)

fimalgoritmo
```

g) Elabore um algoritmo que receba do usuário “n” números inteiros (quanto ele quiser). O algoritmo deverá ler os n números e devolver:

- O maior e o menor número.
- A média dos números.
- A somatória desses números.
- A produtória desses números (o resultado da multiplicação de todos eles).
- A quantidade de números positivos e a quantidade de números negativos.
- A quantidade de números pares e a quantidade de números ímpares.

```
algoritmo "Exercicio_Gerais_Capítulo_3_g"

var
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  i: inteiro;
  numero: inteiro;
  valor: inteiro;
  soma: inteiro;
  menor: inteiro;
  maior: inteiro;
  produtoria: inteiro;
  pares: inteiro;
  impares: inteiro;
  positivos: inteiro;
  negativos: inteiro;


inicio

  maior := -999999999
  menor := 999999999
  produtoria := 1

  escreva("Informe a quantidade de números que deSEja informar: ")
  leia(numero)

  PARA i de 1 ATE numero FACA

    escreva ("Informe um número: ")
    leia (valor)

    soma := soma + valor
    produtoria := produtoria * valor
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    SE valor > maior ENTÃO
      maior := valor
    FIMSE

    SE valor < menor ENTÃO
      menor := valor
    FIMSE

    SE (valor MOD 2 = 0)  ENTÃO
      pares := pares + 1
    FIMSE

    SE (valor MOD 2 = 1) ENTÃO
      impares := impares + 1
    FIMSE

    SE valor >= 0 ENTÃO
      positivos := positivos + 1
    FIMSE

    SE valor < 0 ENTÃO
      negativos := negativos + 1
    FIMSE

  FIMPARA

  escreval ("A soma é : ", soma)
  escreval ("A produtória é : ", produtoria)
  escreval ("O maior valor é : ", maior)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  escreval ("O menor valor é : ", menor)
  escreval ("A quantidade de números pares digitada foi : ", pares)
  escreval ("A quantidade de números ímpares digitada foi : ", impares)
  escreval ("A quantidade de números positivos digitada foi : ", positivos)
  escreval ("A quantidade de números negativos digitada foi : ", negativos)

fimalgoritmo
```

h) Dizemos que um número natural n é palíndromo se:
o 1º algarismo de n é igual ao seu último algarismo;
o 2º algarismo de n é igual ao penúltimo algarismo;
e assim sucessivamente.

Exemplos:

- 567765 e 32423 são palíndromos.
- 567675 não é palíndromo.

Elabore um algoritmo que receba um número inteiro "n" e descubra se "n" é palíndromo ou não.

```
algoritmo "Exercicio_Gerais_Capítulo_3_h"

var

  numero: inteiro;
  aux: inteiro;
  reverso: inteiro;

inicio

  escreva("Informe um número natural: ")
  leia(numero)
  aux := numero
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  ENQUANTO aux <> 0 FAÇA
    reverso := reverso * 10 + aux MOD 10
    aux := INT(aux / 10)
  FIMENQUANTO

  SE reverso = numero ENTÃO
    escreva (numero, " é palíndromo")
  SENÃO
    escreva (numero, " não é palíndromo")
  FIMSE

fimalgoritmo
```

i) Elabore um algoritmo que recebe do usuário dois números “p” e “q” inteiros e positivos, calcule e exiba o MMC(p, q) (mínimo múltiplo comum de p e q).

```
algoritmo "Exercicio_Gerais_Capítulo_3_i"

var

  mmc: inteiro;
  a: inteiro;
  b: inteiro;
  c: inteiro;
  d: inteiro;
  r: inteiro;

inicio

  escreva("Informe um número inteiro e positivo: ")
  leia(a)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    c := a
    escreva("Informe outro número inteiro e positivo: ")
    leia(b)
    d := b

    ENQUANTO b <> 0 FAÇA
      r := a MOD b
      a := b
      b := r
    FIMENQUANTO

    mmc := c * INT((d / a))

      escreva ("O MMC de ", c, " e ", d, " é ", mmc)

  fimalgoritmo
```

j) Elabore um algoritmo que imprima todos os números primos no intervalo de 1 a 100.000.

```
algoritmo "Exercicio_Gerais_Capítulo_3_j"

var

  i: inteiro;
  j: inteiro;
  numero: inteiro;
  contador: inteiro;

inicio

  PARA i de 1 ATE 100000 FACA
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade n.ferreira.tavares@avanade.com

```
    numero := i
    contador := 0
    PARA j de 1 ATE numero FACA
      SE numero MOD j = 0 ENTÃO
        contador := contador + 1
      FIMSE
    FIMPARA
    SE contador = 2 ENTÃO
      escreval (numero)
    FIMSE
  FIMPARA

fimalgoritmo
```

k) Dois números inteiros positivos são chamados primos entre si se o MDC entre eles for 1. Elabore um algoritmo que receba do usuário um número inteiro “n”, calcule e imprima “os números primos entre si” entre esse número e números no intervalo entre 1 e 100.

```
algoritmo "Exercicio_Gerais_Capítulo_3_k"

var

  i: inteiro;
  numero: inteiro;
  valor: inteiro;
  resto: inteiro;
  maior: inteiro;
  menor: inteiro;

inicio
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    escreva ("Entre com um número positivo menor que 100: ")
    leia (numero)

    PARA i de numero+1 ATE 100 FACA
      menor := numero
      maior := i
      valor := i
      ENQUANTO (maior MOD menor <> 0) FAÇA
        resto := maior MOD menor
        maior := menor
        menor := resto
      FIMENQUANTO
      SE menor = 1 ENTÃO
        escreval (numero, " e ", valor, " são primos entre sí")
      FIMSE
    FIMPARA

fimalgoritmo
```

l) Elabore um algoritmo que receba do usuário “n” números, calcule e imprima a soma somente daqueles que pertencerem a Progressão Aritmética 4,7,10,13,16,19...

```
algoritmo "Exercicio_Gerais_Capítulo_3_l"

var

  i: inteiro;
  j: inteiro;
  numero: inteiro;
  valor: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
inicio

  escreva ("Quantos números serão digitados: ")
  leia (numero)

  PARA i de 1 ATE numero FACA
    escreva ("Digite o ", i, "º número :")
    leia (valor)
    SE valor > 3 ENTÃO
      PARA j de 4 ATE valor PASSO 3 FACA
        SE valor = j ENTÃO
          escreval ("O número ", valor, " faz parte da sequencia 4,7,10,13,16,19...")
        FIMSE
      FIMPARA
    FIMSE
  FIMPARA

fimalgoritmo
```

m) Elabore um algoritmo que receba do usuário um número inteiro “n” qualquer, calcule e imprima os números de Fibonacci inferiores ao número informado.

```
algoritmo "Exercicio_Gerais_Capítulo_3_m"

var

  v1: inteiro;
  v2: inteiro;
  v3: inteiro;
  contador: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  qtd: inteiro;
inicio

  v1 <- 1
  v2 <- 1
  v3 <- 1
  Escreva("Digite quantos numeros na sequencia Fibonacci: ")
  Leia(qtd)
  Escreval(v3)
  para contador de 1 ate qtd faca
    Escreval(v3)
    v3 <- v1 + v3
    v1 <- v2
    v2 <- v3

  fimpara

fimalgoritmo
```

n) Elabore um algoritmo que imprima todos os pares de números primos entre si no intervalo de 2 a 100. Por exemplo: (2,3), (2,5), (2,7),...,(2, 99); (3,4), (3,5),...(3,100); (4,5), (4,6),...

```
algoritmo "Exercicio_Gerais_Capítulo_3_n"

var

  i: inteiro;
  j: inteiro;
  numero: inteiro;
  valor: inteiro;
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  resto: inteiro;
  maior: inteiro;
  menor: inteiro;

inicio

  para j de 2 ate 100 faca
    numero := j
    PARA i de numero+1 ATE 100 FACA
      menor := numero
      maior := i
      valor := i
      ENQUANTO (maior MOD menor <> 0) FAÇA
        resto := maior MOD menor
        maior := menor
        menor := resto
      FIMENQUANTO
      SE menor = 1 ENTÃO
        escreval (numero, " e ", valor, " são primos entre sí")
      FIMSE
    FIMPARA
  FIMPARA

Fimalgoritmo
```

o) Elabore um algoritmo que simule o “lápis tabuada”: o usuário informa um número inteiro qualquer, calcule e exiba os resultados das multiplicações entre 1 e 10.

```
algoritmo "Exercicio_Gerais_Capítulo_3_o"

var
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    i: inteiro;
    numero: inteiro;

inicio

    ESCREVA ("Entre com um número entre 1 e 10: ")
    leia (numero)

    PARA i de 1 ATE 10 FACA
       escreval (numero, " X ", i, " = ", numero * i )
    FIMPARA

fimalgoritmo
```

p) Considere a PG (progressão geométrica) 1,2,4,8,16,... Elabore um algoritmo que solicita ao usuário um número inteiro e positivo “n”, calcular e exiba seus “n” primeiros termos e sua somatória de todos eles.

```
algoritmo "Exercicio_Gerais_Capítulo_3_p"

var

    i: inteiro;
    numero: inteiro;
    soma: inteiro;
    pg: inteiro;

inicio

    ESCREVA ("Entre o número de termos da PG: ")
    leia (numero)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  pg := 1;

  PARA i de 1 ATE numero FACA
    se numero - i <> 0 ENTAO
      escreva (pg, " + ")
    SENAO
      escreva (pg)
    fimse
    soma := soma + pg
    pg := pg * 2
  FIMPARA

  escreva (" = ", soma)

fimalgoritmo
```

q) Elabore um algoritmo que calcule e exiba o seguinte cálculo: 1 – ½ + ¼ - 1/6 + 1/8 - ... + 1/200

```
algoritmo "Exercicio_Gerais_Capítulo_3_q"

var

  i: inteiro;
  soma: real;

inicio

  PARA i de 1 ATE 200 FACA
    SE i MOD 2 = 1 ENTAO
      soma := soma + 1/i
    SENAO
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
        soma := soma - 1/i
      FIMSE
    FIMPARA

    escreva (soma)

fimalgoritmo
```

r) Considere-se que um único casal de coelhos depois de dois meses de vida e, a partir daí, produz um novo casal de coelhos a cada mês. Se os coelhos nunca morrem, a quantidade de casal de coelhos após “n” meses é dada pelo n-ésimo termo da série: $F_n = F_{n-2} + F_{n-1}$, onde n >= 2 e $F_0 = 1$ e $F_1 = 1$

Elabore um algoritmo capaz de calcular a quantidade de casais de coelhos após “n” meses, sendo “n” um número inteiro e positivo fornecido pelo usuário.

```
algoritmo "Exercicio_Gerais_Capítulo_3_R"

var

  v1: inteiro;
  v2: inteiro;
  v3: inteiro;
  i: inteiro;
  qtd: inteiro;
  soma: inteiro;

inicio

  v1 := 1
  v2 := 1
  v3 := 1
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade n.ferreira.tavares@avanade.com

```
soma := v3
Escreva("Digite quantidade de meses: ")
Leia(qtd)
para i de 1 ate qtd faca
   soma := soma + v3
   v3 := v1 + v3
   v1 := v2
   v2 := v3
fimpara

escreva ("A quantidade de coelhos será: ", soma)

    fimalgoritmo
```

4.12 Laços usando funções

Temos inúmeras situações em que podemos usar funções para resolver algum problema específico e, toda vez que precisarmos resolver esse mesmo problema, basta invocarmos a função originalmente construída para tal finalidade.

Por exemplo, já criamos vários algoritmos para identificação de todo tipo de coisa. Que tal criarmos agora um algoritmo, ou melhor, uma função que devolva ao programa chamador “1” se um número que a função receber for primo ou “0”, em caso contrário?

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo funcao_primo;
variáveis
  a : inteiro;
  primo : lógico;
fim-variáveis
início
     imprima("Digite um Número: ");
    a := leia();
    se E_Primo(a) = 1 então
        imprima(a," não é primo!");
      senão
        imprima(a," é primo!");
    fim-se
fim
função E_Primo(n: inteiro): inteiro
    primo, i : inteiro;
início
      primo := 0;
      para i de 2 até n - 1 faça
        se n % i = 0 então
              primo := 1;
        fim-se
    fim-para
    retorne primo;
fim
```

Figura 4.34 – Algoritmo do número primo, transformado em função.
Fonte: Elaborado pelo autor (2015)

Como podemos notar, a função tem como grande mérito possibilitar o reaproveitamento de seu “código fonte”. Na programação das atuais linguagens de computador, isso é usualmente chamado de método. Uma vez criada uma função, podemos usá-la sempre com a certeza de que seus resultados serão corretos.

4.12.1 Exercícios

a) Elabore uma função que receba um número inteiro informado pelo usuário e retorne “1” se este número fizer parte da série de Fibonacci ou “0” em caso contrário.

   algoritmo "Exercicio_Funções_a"

   var

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  numero: inteiro;

funcao fibo (n: inteiro): inteiro

var

  i: inteiro;
  v1: inteiro;
  v2: inteiro;
  v3: inteiro;
  sim: inteiro;

inicio

  v1 := 1
  v2 := 1
  v3 := 1
  sim := 0

  PARA i de 1 ate n FACA
    v3 := v1 + v3
    v1 := v2
    v2 := v3
    SE v3 = n ENTAO
      sim:= 1
    FIMSE
  FIMPARA

  RETORNE sim

fimfuncao
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
inicio

  ESCREVA("Digite um número inteiro positivo: ")
  LEIA(numero)
  SE FIBO(numero) = 1 ENTAO
    escreva ("O numero ", numero, " faz parte da sequencia de fibonacci!")
  SENAO
    escreva ("O numero ", numero, " não faz parte da sequencia de fibonacci!")
  fimse

fimalgoritmo
```

b) Elabore uma função que receba um número inteiro informado pelo usuário e retorne “1” se este número for palíndromo ou “0” em caso contrário.

```
algoritmo "Exercicio_Funções_b"

var

  numero: inteiro;

funcao PALINDROMO (n: inteiro): inteiro

var

  aux: inteiro;
  reverso: inteiro;

inicio

  aux := n
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  ENQUANTO aux <> 0 FAÇA
    reverso := reverso * 10 + aux MOD 10
    aux := INT(aux / 10)
  FIMENQUANTO

  SE reverso = n ENTÃO
    retorne 1
  SENÃO
    retorne 0
  FIMSE

fimfuncao

inicio

  escreva("Informe um número natural: ")
  leia(numero)

  SE PALINDROMO(numero) = 1 ENTAO
    escreva ("O numero ", numero, " é palindromo")
  SENAO
    escreva ("O numero ", numero, " não é palindromo")
  fimse

fimalgoritmo
```

c) Elabore uma função que receba um número inteiro informado pelo usuário e retorne “1” se este número fizer parte da Progressão Aritmética 4, 7, 10, 13, 16, 19, ou “0” em caso contrário.

```
algoritmo "Exercicio_Funções_c"
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
var

  numero: inteiro;

funcao SEQ (valor: inteiro): inteiro

var

  j: inteiro;

inicio

  SE valor > 3 ENTÃO
    PARA j de 4 ATE valor PASSO 3 FACA
      SE valor = j ENTÃO
        retorne 1
      SENAO
        retorne 0
      FIMSE
    FIMPARA
  SENAO
    retorne 0
  FIMSE

fimfuncao

inicio

  escreva("Informe um número natural: ")
  leia(numero)
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
    SE SEQ(numero) = 1 ENTAO
      escreva ("O numero ", numero, " faz parte da sequencia 4, 7, 10, 13, 16, 19")
    SENAO
      escreva ("O numero ", numero, " não faz parte da sequencia 4, 7, 10, 13, 16, 19")
    fimse
fimalgoritmo
```

4.13 Funções recursivas

Outra importante aplicação de funções está na construção de algoritmos que possam se autochamar, criando a impressão da existência de laços infinitos.

4.13.1 Funções recursivas - definição

Antes de qualquer coisa, vamos entender o que é Recursividade, com exatidão.

4.13.1.1 Definição

Um programa recursivo é um programa que chama a si mesmo, direta ou indiretamente.

Trata-se de um conceito importante, pois permite que *conjuntos* infinitos sejam alcançados a partir de *comandos* finitos.

- Vantagens:
    - Redução do tamanho do código fonte.
    - Permite descrever algoritmos de forma mais clara e concisa.
- Desvantagens:
    - Redução do desempenho de execução devido ao tempo para gerenciamento de chamadas.
    - Dificuldades na depuração de programas recursivos, especialmente se a recursão for muito profunda.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Mas, e como a recursividade funciona na memória do computador?

Basicamente, segue o processo descrito adiante?

- Usa-se uma pilha para armazenar os dados usados em cada chamada de um procedimento / função que não terminou.
- Todos os dados não globais são armazenados na pilha, informando o resultado corrente.
- Quando uma ativação anterior prossegue, os dados da pilha são recuperados.

Colocando-se esses procedimentos numa espécie de loop, mas que é controlado pela própria função, tem-se a recursividade!

4.13.2 Exemplo: fatorial

Vamos pensar no clássico problema do cálculo do fatorial de um número qualquer. Podemos fazer isso de duas formas bastante distintas:

- Definição de uma Função Fatorial:
  a) Não Recursiva:
     - N! = 1, para N=0;
     - N! = 1 x 2 x 3 x ... x N, para N>=1;
  b) Recursiva:
     - N! = 1, para N=0;
     - N! = N x (N – 1), para N>=1;

Primeiramente, vamos implementar a função fatorial de forma não recursiva, isto é, um programa chamador passa a função um número e recebe como retorno seu fatorial, todavia a função faz o cálculo desse valor de forma convencional.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade n.ferreira.tavares@avanade.com

```
algoritmo funcao_fatorial_convencional;
início
fim
função fatorial (n: inteiro): real
    i : inteiro;
    result: real;
início
    result := 1;
    para i de 2 até n faça
        result := result * i;
    fim-para
    retorne result;
fim
```

Figura 4.35 – Algoritmo de cálculo fatorial, da forma convencional.
Fonte: Elaborado pelo autor (2015)

Podemos observar que a função recebe uma variável “n” e, através de cálculo repetitivo, devolve ao programa principal o valor de seu fatorial.

Por outro lado, poderíamos pensar numa função que se “auto chamasse”. Observemos a animação que retrata uma função “recursiva”.

```
algoritmo funcao_fatorial_recursiva;
início
fim
função fatorial (n: inteiro): real
início
    se n = 0 então
        retorne 1;
    senão
        retorne n * fatorial(n-1);
    fim-se
fim
```

Figura 3.36 – Algoritmo de cálculo fatorial utilizando função recursiva .
Fonte: Elaborado pelo autor (2015)

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Notemos que a função fatorial se “auto chama” inúmeras vezes, até que o número chamado seja 0 (zero). Quando esse número é chamado, a função retorna o valor 1 (um) e resolve o processo que, até então, estava pendente.

Isso “dispara” a solução de todas as outras “pendências” existentes. Ou seja, só quando determino 0! (que vale 1), poderei calcular 1!, 2!, 3! e assim por diante.

Note, ainda, que os procedimentos recursivos introduzem a possibilidade de iterações que podem não terminar: existe a necessidade de considerar o problema de *terminação*!

É fundamental que a chamada recursiva a um procedimento qualquer “P” esteja sujeita a uma condição específica “A”, a qual se torna satisfeita em algum momento da computação.

- Exemplo: Se não existisse a condição *n=0*, quando o procedimento “Fatorial” terminaria? Loop Eterno!

Condição de terminação:

- Permite que o procedimento deixe de ser executado.
- O procedimento deve ter pelo menos um caso básico para cada caso recursivo, o que significa a finalização do procedimento.

4.13.3 Exemplo: potenciação

Vamos agora criar um algoritmo recursivo para elevar um número a uma potência inteira não negativa.

Algebricamente falando, desejamos calcular a seguinte expressão:

$$x^n = \begin{cases} 1 & \text{se } n = 0 \\ x \cdot x^{n-1} & \text{se } n > 0 \end{cases}$$

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo funcao_potencia_recursiva;
início
fim
função potencia( x, n : inteiro):inteiro
início
  se n = 0 então
     retorne 1 ;
  senão
      se n = 1 então
          retorne x;
      senão
          retorne x * potencia( x, n - 1 );
      fim-se
    fim-se
fim
```

Figura 4.37 – Algoritmo de cálculo exponencial utilizando função recursiva
Fonte: Elaborado pelo autor (2015)

Notemos que nossa função tem dois parâmetros: “x” que representa o número cuja potenciação queremos obter e “n” que indica a qual potência devemos elevar “x”. Observar que temos uma condição de término. Quando “n” for igual a 0 (zero), então obtemos como retorno o valor 1 (de fato, qualquer número elevado a zero resulta em um).

Já para qualquer outro valor sabemos que basta multiplicar o número por ele mesmo, elevado a sua potência menos o valor um. Por isso, quando fatorial se “autochama”, temos a passagem de dois parâmetros, ou seja, o número a ser elevado e a potência decrementada de um.

4.13.4 Exemplo: torre de HANÓI

O problema em questão foi criado pelo matemático francês Edouard Lucas. Inspirado por uma lenda Hindu, que falava de um templo em Bernares, uma cidade santa da Índia, onde existiria uma torre sagrada do bramanismo, cuja função era melhorar a disciplina mental dos monges jovens.

A lenda dizia que, no início dos tempos, foi dada aos monges de um templo uma pilha de 64 discos de ouro, dispostos em uma haste, de forma que cada disco de cima fosse menor que o de baixo.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

A atribuição que os monges receberam foi transferir a torre, formada pelos discos, de uma haste para outra, usando a terceira como auxiliar com as restrições de movimentar um disco por vez e de nunca colocar um disco maior sobre um menor.

Os monges deveriam trabalhar com eficiência noite e dia e quando terminassem o trabalho, o templo seria transformado em pó e o mundo acabaria.

Figura 4.38 – Torre de Hanói
Fonte: Banco de imagens Shutterstock (2016)

Simplificando-se a questão para entendimento, fiquemos com três torres e com três discos. O objetivo seria transferir os três discos da torre A para a torre C usando a torre B como auxiliar, sendo que somente o primeiro disco de uma torre pode ser deslocado para outro e nunca um disco maior pode ser posicionado sobre um outro menor.

No site www.scielo.com.br encontramos um “proto-algoritmo” bastante interessante:

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

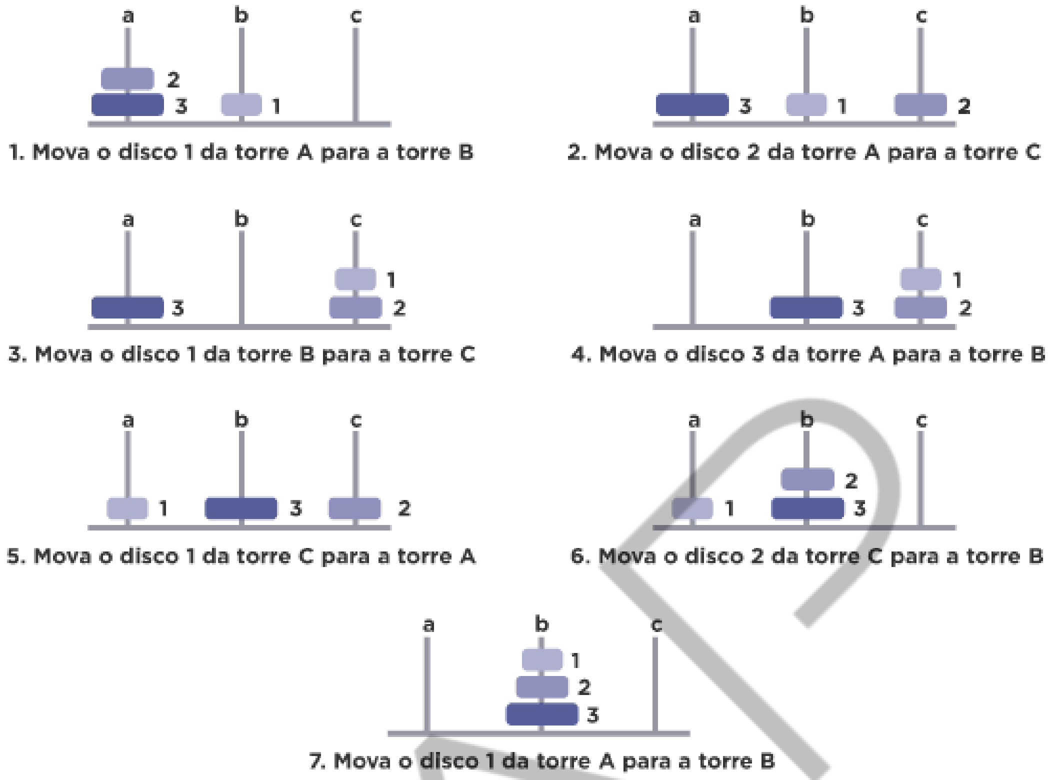


Figura 4.39 – Representação gráfica e em linguagem natural para a Torre Hanoi
Fonte: Adaptado por FIAP (2015)

Para solucionar o problema das torres de Hanoi com uma recursão, devemos considerar que os "n" discos devam ser transferidos.

Dessa maneira podemos "dividir" o problema em dois casos mais simples, que irão mover os "n" discos.

O primeiro, através da solução trivial (quando um disco tiver que ser movido entre as torres A e C) e o segundo através da solução geral (uma solução para "n" discos em termos de "n-1").

Com três discos é possível entender a estratégia recursiva, admitindo mover 2 discos da torre A para a torre B, usando a torre C, que nos levaria...

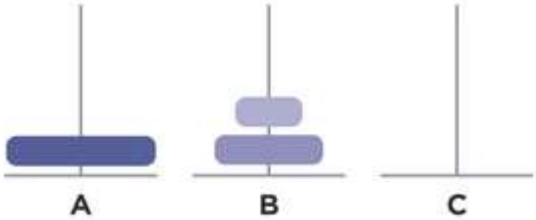


Figura 4.40 – Exemplo de estratégia recursiva
Fonte: Adaptado por FIAP (2015)

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

Depois poderíamos mover o disco maior diretamente da torre A para a torre C.

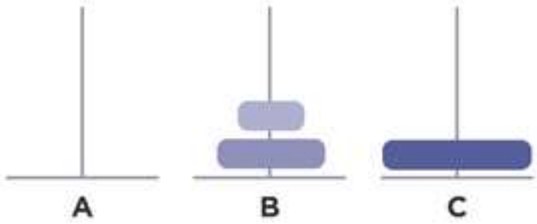


Figura 4.41 – Mudança do disco maior diretamente da torre A para a torre C
Fonte: Adaptado por FIAP (2015)

Novamente é possível aplicar a solução recursiva para mover os dois discos da torre B para a torre C, agora usando a torre A.

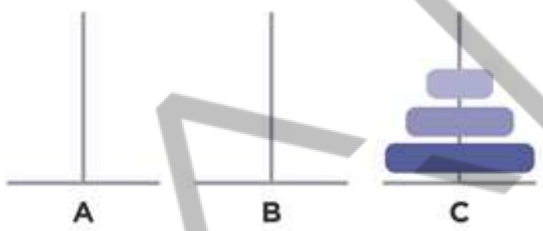


Figura 4.42 – Mudança dos dois discos da torre B para a torre C
Fonte: Adaptado por FIAP (2015)

Basicamente ocorreu o seguinte:

a) Se n=1 então transfira o disco de A para C, parando.

b) Senão:

- Transfira n-1 discos de A para B usando C como auxiliar.
- Transfira o último disco de A para C.
- Transfira n-1 discos de B para C usando A como auxiliar

c) Que nos leva...

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

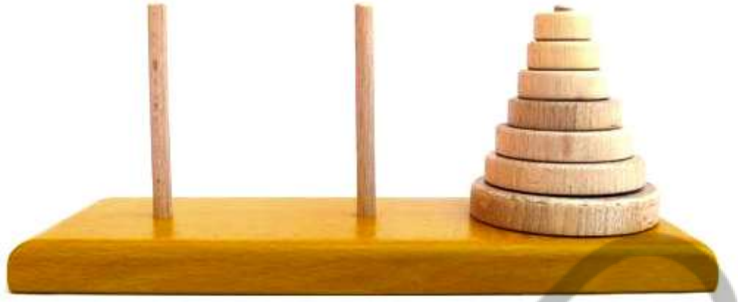


Figura 4.43 – Torre de Hanói
Fonte: Banco de imagens Shutterstock (2016)

4.14 Exercícios clássicos

Os seguintes exercícios, chamados aqui de clássicos, são usualmente encontrados em provas ou testes realizados em empresas. Alguns foram tratados anteriormente nesse material ou tratamos de casos muito parecidos. De qualquer forma, seguem os "clássicos", fechando esse capítulo.

1. Elabore um algoritmo que calcule o fatorial de um número, utilizando uma função. Chame a função algumas vezes para testá-la.

```
algoritmo "Exercicio_Clássicos_1"

var

entrada: inteiro;

funcao FATO (numero: inteiro): inteiro

 var
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
  i: inteiro;
  fat: inteiro;

inicio

  fat := 1

  PARA i de 1 ATE numero FACA
    fat := fat * i
  FIMPARA

  retorne fat

fimfuncao

inicio

  escreva("Informe um número inteiro: ")
  leia(entrada)

  escreval("Fatorial de ", entrada, " é ", FATO(entrada))


fimalgoritmo
```

2. Elabore um algoritmo que possua uma função que receba como parâmetros dois números e devolva a soma dos primos existentes entre ambos. Chame a função algumas vezes para testá-la.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo "Exercicio_Clássicos_2"

var

  entrada: inteiro;
  ultimo: inteiro;
  j: inteiro;

funcao PRIMO (numero: inteiro): inteiro

var

  i: inteiro;
  contador: inteiro;

inicio

  PARA i de 1 ATE numero FACA
    SE numero MOD i = 0 ENTÃO
      contador := contador + 1
    FIMSE
  FIMPARA

  SE contador = 2 ENTÃO
    retorne 1
  SENÃO
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
        retorne 0
      FIMSE

   fimfuncao

   inicio

      escreva("Informe um número inteiro: ")
      leia(entrada)

      escreva("Informe um número inteiro maior que o anterior: ")
      leia(ultimo)

      PARA j DE entrada ATE ultimo FACA
        SE PRIMO(j) = 1 ENTÃO
          escreval (j)
        FIMSE
      FIMPARA

   fimalgoritmo
```

3. Elabore um algoritmo que possua uma função que receba como parâmetros dois números e devolva a soma dos números de Fibonacci existentes entre ambos. Chame a função algumas vezes para testá-la.

```
   algoritmo "Exercicio_Clássicos_3"
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
var

   entrada: inteiro;
   ultimo: inteiro;
   j: inteiro;

funcao fibo (n: inteiro): inteiro

var

   i: inteiro;
   v1: inteiro;
   v2: inteiro;
   v3: inteiro;

inicio

   v1 := 1
   v2 := 1
   v3 := 1

   PARA i de 1 ate n FACA
      v3 := v1 + v3
      v1 := v2
      v2 := v3
      SE v3 = n ENTAO
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
      retorne 1
    SENAO
      retorne 0
    FIMSE
  FIMPARA

fimfuncao

inicio

  escreva("Informe um número inteiro: ")
  leia(entrada)

  escreva("Informe um número inteiro maior que o anterior: ")
  leia(ultimo)

  PARA j DE entrada ATE ultimo FACA
    SE FIBO(j) = 1 ENTÃO
      escreval (j)
    FIMSE
  FIMPARA

fimalgoritmo
```

4. Elabore um algoritmo que possua uma função que receba o total de números e imprima os “n” primeiros termos da PA 4, 7, 10, 13, 16, 19, ... Chame a função algumas vezes para testá-la.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
algoritmo "Exercicio_Clássicos_4"

var

  numero: inteiro;

funcao SEQ (valor: inteiro): inteiro

var

  j: inteiro;

inicio

  SE valor > 3 ENTÃO
    PARA j de 4 ATE valor PASSO 3 FACA
      SE valor = j ENTÃO
        retorne 1
      SENAO
        retorne 0
      FIMSE
    FIMPARA
  SENAO
    retorne 0
  FIMSE

fimfuncao

inicio

  escreva("Informe um número natural: ")
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   leia(numero)

   SE SEQ(numero) = 1 ENTAO
      escreva ("O numero ", numero, " faz parte da sequencia 4, 7, 10, 13, 16, 19")
   SENAO
      escreva ("O numero ", numero, " não faz parte da sequencia 4, 7, 10, 13, 16, 19")
   fimse
fimalgoritmo
```

5. Elabore um algoritmo que possua uma função que receba como parâmetros dois números e apresente os números palíndromos existentes entre ambos. Chame a função algumas vezes para testá-la.

```
    algoritmo "Exercicio_Clássicos_5"

var

   numero: inteiro;

funcao PALINDROMO (n: inteiro): inteiro

var

   aux: inteiro;
   reverso: inteiro;

inicio

   aux := n
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

```
   ENQUANTO aux <> 0 FAÇA
      reverso := reverso * 10 + aux MOD 10
      aux := INT(aux / 10)
   FIMENQUANTO

   SE reverso = n ENTÃO
      retorne 1
   SENÃO
      retorne 0
   FIMSE

fimfuncao

inicio

   escreva("Informe um número natural: ")
   leia(numero)

   SE PALINDROMO(numero) = 1 ENTAO
      escreva ("O numero ", numero, " é palindromo")
   SENAO
      escreva ("O numero ", numero, " não é palindromo")
   fimse

fimalgoritmo
```

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com

REFERÊNCIAS

ENCYCLOPEDIA and history of programming languages. [s.d.]. Disponível em: <http://www.scriptol.org/>. Acesso em: 14 jan. 2011.

FEOFILOFF, Paulo. **Algoritmos em Linguagem C.** Rio de Janeiro: Campus, 2009.

FORBELLONE, André L.V.; EBERSPACHER, Henri F. **Construção de Algoritmos e Estruturas de Dados**. São Paulo: Pearson Prentice Hall, 2010.

FURGERI, Sérgio. **Java 2, Ensino Didático**. São Paulo: Érica, 2002.

GANE, Chris e SARSON, Trish. **Análise Estruturada de Sistemas**. São Paulo: LTC- Livros Técnicos e Científicos, 1983.

GONDO, Eduardo. **Apostila: Notas de Aula**. São Paulo, 2008.

LATORE, Robert. **Aprenda em 24 horas Estrutura de Dados e Algoritmos**. Rio de Janeiro: Campus, 1999.

MANZANO, José A.N.G. e OLIVEIRA, Jayr F. **Algoritmos:** Lógica para o Desenvolvimento de Programação. 23.ed. São Paulo: Érica, 2010.

PIVA JUNIOR, Dilermando et al. **Algoritmos e Programação de Computadores.** Rio de Janeiro: Campus, 2012.

PUGA, Sandra; RISSETTI, Gerson. **Lógica de Programação e Estrutura de Dados**. São Paulo: Pearson Prentice Hall, 2009.

ROCHA, Antonio Adrego. **Estrutura de Dados e Algoritmos em Java**. Lisboa: FCA- Editora de Informática, 2011.

RODRIGUES, Rita. Apostila: Notas de Aula. 2008.

SALVETTI, Dirceu Douglas e BARBOSA, Lisbete Madsen. **Algoritmos**. São Paulo: Makron Books, 1998.

SCHILDT, Herbert. **Linguagem C – Guia Prático**. São Paulo: McGraw Hill, 1989.

WOOD, Steve. **Turbo Pascal – Guia do Usuário**. São Paulo: McGraw Hill, 1987.

ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Pascal e linguagem C.** São Paulo: Pioneira, 1999.

PDF exclusivo para Nathallye Thalya Regina Ferreira Tavares - nathallyetavares.avanade
n.ferreira.tavares@avanade.com