# Introdução à Computação

Rosane Minghim e Guilherme P. Telles

9 de Agosto de 2012

# Capítulo 3

# Estruturas de Controle

Os algoritmos apresentados até agora (à excessão de alguns exemplos genéricos) permitiam apenas um fluxo de execução seqüencial, ou seja, uma vez que o primeiro comando é executado, todos os demais são executados em seqüência, até o último. Na maioria das vezes, no entanto, comandos são executados apenas em determinadas condições, ou então precisam ser repetidos várias vezes, mudando a seqüencia direta de execução apresentada até agora. Neste capítulo são apresentadas algumas estruturas usadas em algoritmos, as quais chamamos de **estruturas de controle**. Essas estruturas modificam o fluxo de execução do algoritmo, permitindo escolher uma ou mais alternativas de comandos ou permitindo repetir um conjunto de comandos, alterando assim o a seqüência de um algoritmo.

## 3.1 Escolhas

Em um algoritmo podemos fazer uma escolha baseada no resultado de uma lógica, isto é, de uma condição. Uma escolha simples possui o seguinte formato:

```
se condição então
  comandos
fim se
```

Nesse caso, apenas se a condição for verdadeira o bloco de comandos será executado. Se a condição for falsa, o fluxo do algoritmo é desviado da condição da cláusula `se` para o primeiro comando após a cláusula `fim se`.

O trecho de algoritmo abaixo calcula o valor da taxa mensal paga por um cliente de banco, de tal forma que o cliente recebe um desconto de 100% se

for da categoria 1.

**Exemplo 3.1**

```
constante
  FATOR_CATEGORIA_1 = 1,0
  TAXA_BÁSICA = 20,00

variável
  categoria: inteiro
  taxa: real

leia(categoria)

taxa ← taxa_básica
se categoria = 1 então
  taxa ← taxa - (TAXA_BÁSICA * FATOR_CATEGORIA_1)
fim se

escreva(taxa)
```

Outra forma possível para estrutura de controle baseada em escolha é a escolha composta, que possui o seguinte formato:

```
se condição então
  comandos 1
senão
  comandos 2
fim se
```

Nesse caso, se a condição for verdadeira o bloco `comandos 1` será executado e se a condição for falsa, o bloco `comandos 2` será executado. Nos dois casos, o primeiro comando a ser executado após a execução do bloco correspondente é aquele que aparece no algoritmo logo após a cláusula `fim se`.

O trecho de algoritmo abaixo calcula o valor da taxa mensal paga por um cliente de banco, de tal forma que o cliente recebe um desconto de 100% se for da categoria 1 ou de 20% se não for dessa categoria.

**Exemplo 3.2**

```
constante
  FATOR_CATEGORIA_1 = 1,0
  FATOR_GERAL = 0,2
```

```
  TAXA_BÁSICA = 20,00

variável
  categoria: inteiro
  taxa: real

leia(categoria)

taxa ← taxa_básica
se categoria = 1 então
  taxa ← taxa - (TAXA_BÁSICA * FATOR_CATEGORIA_1)
senão
  taxa ← taxa - (TAXA_BÁSICA * FATOR_GERAL)
fim se

escreva(taxa)
```

**Sugestão 5** *A partir deste ponto, o leitor deve tentar desenvolver os algoritmos apresentados nos exemplos antes de ler a resposta. Depois de ler a resposta, o leitor ainda deve tentar melhorar o algoritmo, procurando uma solução melhor para o problema ou um algoritmo otimizado ou mais claro.*

Estruturas de controle podem ser 'aninhadas', isto é, os blocos de comandos subordinados a condições podem conter outras escolhas, conforme ilustrado no próximo exemplo, que obtém o maior entre três valores numéricos.

**Exemplo 3.3** *O maior entre três números*

```
1  Algoritmo maiordetres
2
3    variável
4      maior: inteiro
5      numero1, numero2, numero3: inteiro
6
7    leia(numero1, numero2, numero3)
8    se numero1>numero2 então
9      se (numero2 ≥ numero3) então
10       maior ← numero1
11     senão
12       se numero1>numero3 então
13         maior ← numero1
```

```
14        senão
15          maior ← numero3
16        fim se
17      fim se
18    senão
19      se (numero2 ≥ numero3) então
20        maior ← numero2
21      senão
22        maior ← numero3
23      fim se
24    fim se
25  fim
```

No algoritmo acima o comando da linha 10 será executado se ambas as condições da linha 8 e 9 forem verdadeiras, e o da linha 13 será executado se a condição da linha 8 for verdadeira e a da linha 9 for falsa. Observe sob que combinação de condições serão executados os comandos das linhas 15, 20 e 22.

**Exercício Sugerido 1** *Com base no algoritmo acima desenvolver um algoritmo para obter o* **menor** *entre três valores numéricos.*

O exemplo adicional abaixo apresenta um algoritmo para, dados dois times de futebol (cada time identificado por um número inteiro), seus pontos ganhos e seu saldo de gols no campeonato, decidir qual dos dois está em melhor colocação (armazenando o resultato na variável `ganhador`). O resultado deve ser impresso. A regra diz que está na frente no campeonato o time que tiver mais pontos ganhos, com desempate pelo saldo de gols.

**Exemplo 3.4**

```
Algoritmo times

  variável
    time1,pontos_ganhos1, saldo_de_gols1: inteiro
    time2, pontos_ganhos2, saldo_de_gols2: inteiro
    ganhador: inteiro


  leia (time1, pontos_ganhos1, saldo_de_gols1)
  leia (time2, pontos_ganhos2, saldo_de_gols2)
```

```
  se pontos_ganhos1 = pontos_ganhos2 então
    se saldo_de_gols1 = saldo_de_gols2 então
      ganhador ← 0
    senão
      se saldo_de_gols1 > saldo_de_gols2 então
        ganhador ← time1
      senão
        ganhador ← time2
      fim se
    fim se
  senão
    se pontos_ganhos1 > pontos_ganhos2 então
      ganhador ← time1
    senão
      ganhador ← time2
    fim se
  fim se
  se ganhador = 0 então
    escreva ('Times empatados na classificacao' )
  senão
    escreva (ganhador)
  fim
fim
```

Deve ser observado, do exemplo acima, que a impressão do resultado ficou isolada no final do algoritmo. Embora o efeito imediato de imprimir no lugar da atribuição fosse o mesmo, um algoritmo com definição de um resultado separadamente da sua impressão é melhor organizado, e mais preparado para futuras modificações.

**Exercício Sugerido 2** *Alterar o programa acima para, ao invés de ler o saldo de gols, ler de fato os gols marcados e sofridos, calculando o saldo. Adicionar também a condição de desempate por número de gols marcados, se os dois outros valores forem iguais para os dois times.*

Abaixo é enunciado e resolvido um exercício que calcula, para um aluno de um curso qualquer, a média final, baseada nas notas de trabalhos e provas, as quais têm pesos diferentes no cálculo da média final.

**Exercício Resolvido 1** *Cálculo da média de um aluno em um curso.*

**Enunciado***: Calcular a média final de um aluno em um curso. No curso, existem três notas de prova e três notas de trabalho, sendo que a menor das notas de prova pode ser substituída pela nota de uma prova substitutiva, que é opcional para o aluno.*

*As notas variam de 0,0 a 10,0. A média de prova (MP) é dada pela média aritmética entre as três notas de prova, e a média de trabalhos (MT) é dada pela média aritmética entre as três notas de trabalho.*

*A média final (MF) é então calculada pela seguinte regra:*

- *MF = 0,7∗MP + 0,3∗MT, se MP e MT forem ambas maiores ou iguais a 5,0.*
- *MF = min(MP,MT) caso contrário.*

*onde: min(a,b) é o menor valor entre a e b.*

*O resultado do programa é uma linha impressa apresentando todas as notas para o aluno, incluindo a média final e as médias intermediárias (prova e trabalho).*

**Observações**

- *Resolva o exercício antes de ver o resultado.*
- *MP, MT e MF são apenas referências usadas neste texto. Não as use como identificadores no programa.*
- *Utilize o algoritmo para cálculo do menor entre três valores sugerido acima (exercício sugerido 1), e use-o para resolver o trecho deste algoritmo que substitui o valor da menor prova pela prova substitutiva.*

***Resolução 1*** *O algoritmo abaixo representa uma possível seqüencia de passos para a solução do problema acima.*

```
Algoritmo médias
  Leia o valor das notas das três provas
  Leia o valor da prova substitutiva
  Obtenha o valor da menor prova
  Calcule a média de provas
  Leia o valor das notas de trabalho
  Calcule a média de trabalhos
  Calcule a média final
  Imprima o resultado
fim
```

*Neste ponto, cada um dos passos acima pode ser desenvolvido separadamente, e depois unido num algoritmo que é, por fim, revisado. O algoritmo abaixo representa uma possível versão para o algoritmo final.*

```
Algoritmo médias

  constante
    MÉDIA_MÍNIMA = 5,0

  variável
    menor_prova: real
    prova1, prova2, prova3, substitutiva: real
    trabalho1, trabalho2, trabalho3: real
    resposta: caracter
    média_prova, média_trabalho, média_final: real

  {Leia o valor das notas das três provas}
  leia (prova1, prova2, prova3)

  leia (resposta)
  se (resposta = 's') ou (resposta = 'S') então
    {Leia o valor da prova substitutiva}
    leia(substitutiva)
    {Obtenha o valor da menor prova}
    se prova1 leq prova2 então
       se (prova2 ≤ prova3)  então
         menor_prova ← prova1
       senão
         se prova1<prova3 então
           menor_prova ← prova1
         senão
           menor_prova ← prova3
         fim se
       fim se
    senão
      se (prova2 ≤ prova3) então
        menor_prova ← prova2
      senão
        menor_prova ← prova3
      fim se
    fim se
```

```
    {Calcule a média de provas - com substitutiva}
    se menor_prova = prova1 então
      média_prova ← (substitutiva + prova2 + prova3)/3
    senão
      se menor_prova = prova2 então
        média_prova ← (prova1 + substitutiva + prova3)/3
      senão
        média_prova ← (prova1 + prova2 + substitutiva)/3
      fim se
    fim se
  senão
    {Calcule a média de provas - sem substitutiva}
    média_prova ← (prova1 + prova2 + prova3)/3
  fim se
  {Leia o valor das notas de trabalho}
  leia(trabalho1,trabalho2,trabalho3)
  {Calcule a média dos trabalhos}
  média_trabalho ← (trabalho1 + trabalho2 + trabalho3)/3
  {Calcule a média final}
  se  (média_trabalho ≥  MÉDIA_MÍNIMA) e (média_prova ≥ MÉDIA_MÍNIMA) então
     média_final ← média_prova*0,7 + média_trabalho*0,3
  senão
    se média_trabalho < média_prova então
      média_final ← média_trabalho
    senão
      média_final ← média_prova
    fim se
  fim se
  {Imprima o resultado}
  escreva(prova1,prova2,prova3,substitutiva,média_prova,média_trabalho,média_fin
fim
```

Um comentário pertinente sobre o algoritmo acima diz respeito à leitura da variável `resposta`. Essa é uma variável tipo caracter, que armazena a resposta à pergunta: 'Existe prova substitutiva (s/n)?'. Como o usuário pode digitar a resposta tanto em maiúscula quanto em minúscula, a estrutura de escolha que segue a leitura deve testar os dois casos.

Também deve ser observado que a estrutura final do algoritmo é ligeiramente diferente daquela do algoritmo de alto nível, isto é, como o cálculo da média das provas é diferente caso haja substitutiva, ele foi dividido em dois :

- {Calcule a média de provas - com substitutiva} e

- {Calcule a média de provas - sem substitutiva}.

os quais foram subordinados ao teste da resposta do usuário sobre a existência de prova substitutiva.

**Exercício Sugerido 3** *Modificar o algoritmo acima para alterar o critério da média de provas para que as provas 1, 2 e 3 tenham pesos 2, 2 e 3 respectivamente.*

**Exercício Sugerido 4** *Alterar o algoritmo do cálculo das médias acima para substituir a menor nota pela nota da prova substitutiva* **apenas** *se a nota da substitutiva for de fato maior que a nota a ser substituída.*

**Exercício Sugerido 5** *Alterar o algoritmo do cálculo das médias acima para atribuir e imprimir um conceito para o aluno. O aluno que obtiver nota entre* $8,5$ *e* $10$ *(inclusive) recebe conceito A. O aluno que obtiver nota acima de* $6,9$ *e abaixo de* $8,5$ *recebe conceito B. O aluno que obtiver nota acima de* $4,9$ *até* $6,9$ *recebe conceito C. O aluno com nota até* $4,9$ *recebe conceito D. O conceito deve ser impresso no final da linha de impressão das notas.*

**Exercício Sugerido 6** *Alterar o algoritmo acima para, antes de atribuir o conceito, arredondar a média para uma casa decimal depois da vírgula. Assim, por exemplo, 4,54 e 4,48 seriam arredondados para 4,5 enquanto que 4,43 ficaria arredondado para 4,4 e 4,55 para 4,6. Dica: usar a função pré-definida* `arredonda(x)` *(ver Seção 2.2.8)*

Em algoritmos, algumas vezes a escolha não é feita entre duas possibilidades apenas (verdadeiro, falso), mas entre várias possibilidades do valor de uma variável ou expressão. Uma estrutura de escolha disponível em pseudocódigo para esses casos é descrita a seguir.

## 3.2 Escolhas Múltiplas

Além das escolhas usando a estrutura **se**, podemos empregar uma outra estrutura que permite escolher uma dentre um conjunto de valores constantes do tipo inteiro ou do tipo caracter:

```
selecione expressão entre
  constante:
    comandos
```

```
  constante:
    comandos
  ...
  constante:
    comandos
  senão
    comandos
fim seleção
```

Usando essa estrutura, a expressão é calculada e os comandos relacionados abaixo da constante com o mesmo valor da expressão serão executados. Se não houver valor igual ao da expressão, os comandos subordinados à palavra `senão` serão executados. A cláusula `senão` é opcional. Se ela não existir e o valor da expressão não for igual a nenhuma constante, nenhum comando da estrutura será executado. Em todo caso, a seleção é exclusiva, isto é, no máximo uma as opções será executada.

No exemplo abaixo, o algoritmo calcula o valor da taxa mensal paga por um cliente de banco dependendo da sua categoria: se for categoria 1, ele recebe 100% de desconto na taxa, se for categoria 2, 80% de desconto, se for 3, 60% de desconto. É dado 20% de desconto para as demais categorias.

**Exemplo 3.5**

```
Algoritmo desconto_taxa
  constante
    FATOR_CATEGORIA_1 = 1,0
    FATOR_CATEGORIA_2 = 0,8
    FATOR_CATEGORIA_3 = 0,6
    FATOR_GERAL = 0,2
    TAXA_BÁSICA = 20,00

  variável
    categoria: inteiro
    taxa: real

  leia(categoria)

  selecione categoria entre
    1:
       taxa ← taxa - (taxa_básica * FATOR_CATEGORIA_1)
    2:
       taxa ← taxa - (taxa_básica * FATOR_CATEGORIA_2)
```

```
    3:
       taxa ← taxa - (taxa_básica * FATOR_CATEGORIA_3)
    senão
      taxa ← taxa - (taxa_básica * FATOR_GERAL)
  fim seleção

  escreva(taxa)
fim
```

Como um exemplo adicional da estrutura de múltipla escolha, o algoritmo a seguir calcula $a^b$, para a e b inteiros, $0 \leq b \leq 4$.

**Exemplo 3.6** *Algoritmo para o cálculo de exponenciação.*

```
Algoritmo expoente

  variável
    base, expoente, resultado: inteiro

  leia(base,expoente)
  resultado ← 0
  selecione expoente entre
  0:
    resultado ← 1
  1:
    resultado ← base
  2:
    resultado ← base*base
  3:
    resultado ← base*base*base
  4:
    resultado ←  base*base*base*base
  senão
    escreva (' Expoentes de 0 a 4 apenas ');
  fim seleção
  escreva(resultado)
fim
```

Abaixo é enunciado e resolvido um exercício que envolve um algoritmo com escolha múltipla.

**Exercício Resolvido 2** *Algoritmo para o cálculo de passagens de ônibus*

**Enunciado** *Calcular o valor de uma cartela de passes de ônibus para um passageiro. Uma cartela pode ter 50 ou 100 passes. Determinados tipos de usuários possuem desconto na compra de passes, de acordo com a tabela abaixo:*

| | |
|---|---|
| *idosos* | *50%* |
| *estudantes* | *50%* |
| *trabalhadores faixa I* | *50%* |
| *trabalhadores faixa II* | *25%* |
| *trabalhadores faixa I e estudante* | *75%* |

***Resolução 2*** *Um algoritmo de alto nível para a solução do problema é muito simples:*

```
Algoritmo passagens
  Leia o valor da passagem sem desconto
  Leia o tipo da cartela de passes
  Leia a categoria do usuário
  Calcule o desconto de acordo com a categoria do passageiro
  Calcule o valor da cartela de passes com base no desconto.
fim
```

*Uma versão para o algoritmo final é dada a seguir.*

```
Algoritmo passagens

  constante
    N_CATEGORIAS = 6
    IDOSO = 1
    ESTUDANTE = 2
    FAIXA_I = 3
    FAIXA_II = 4
    ESTUDANTE_E_FAIXA_I = 5


  tipo
    categorias = 1 até N_CATEGORIAS

  variável
```

```
    passagem_integral: real
    categoria: categorias
    desconto: inteiro
    valor_final: real
    número_de_passes: inteiro


  leia(número_de_passes)
  leia(passagem_integral)
  leia(categoria)
  selecione categoria entre
  IDOSO:
    desconto ← 100
  ESTUDANTE:
    desconto ← 50
  FAIXA_I:
    desconto ← 50
  FAIXA_II:
    desconto ← 25
  ESTUDANTE_E_FAIXA_I:
    desconto ← 75
  senão
    desconto ← 0
  fim seleção
  valor_final ← número_de_passes *
                (passagem_integral - desconto/100 * passagem_integral)
  escreva(valor_final)
fim
```

No algoritmo acima, a categoria 6 é utilizada para representar outros passageiros (isto é, aqueles que não têm desconto). O comando `selecione` encerra as possibilidades de escolha em pseudo-código, que consistem de escolha simples, escolha composta e escolha múltipla.

Em quase todos os exemplos acima, seria interessante que o programa, ao invés de fazer os cálculos para um único caso (um único aluno, um único passageiro, etc.), fizesse para mais pessoas. Para isso é necessário que uma seqüencia de comandos possa ser repetida um número de vezes. Muitas são as situações, em programação, em que existe essa necessidade de repetição de comandos. Seria o caso, por exemplo, do algoritmo de exponenciação dado do Exemplo 3.6. Pode-se perceber que, nesse caso, a multiplicação é repetida um número de vezes. Seria interessante poder expressar esse fato através de

um comando que indique repetição, e generalizar essa repetição para um número variável de multiplicações. Comando que expresse isso é chamado de iteração. As próximas seções apresentam os vários tipos de estrutura existentes em pseudo-código para realizar iteração.

## 3.3 Repetição por condição

Uma das formas de repetir um conjunto de comandos de um algoritmo é subordiná-los a um comando de repetição usando uma estrutura da forma:

```
enquanto condição faça
  comandos
fim enquanto
```

Os comandos serão repetidos zero ou mais vezes, enquanto o valor da condição for verdadeiro. Essa estrutura normalmente é denominada **laço** ou **loop**. Em linhas gerais, a forma de funcionamento de um laço de repetição é assim: a condição da cláusula `enquanto` é testada. Se ela for verdadeira os comandos seguintes são executados em seqüencia como em qualquer algoritmo, até a cláusula `fim enquanto`. O fluxo nesse ponto é desviado de volta para a cláusula `enquanto`. Se a condição agora for falsa (ou quando finalmente for), o fluxo do algoritmo é desviado para o primeiro comando após a cláusula `fim enquanto`. Se a condição ainda for verdadeira, o processo se repete.

A condição pode ser qualquer expressão que resulte em um valor do tipo lógico e pode envolver operadores aritméticos, lógicos, relacionais e resultados de funções.

Por exemplo, poderíamos ler um valor $n$ do teclado e usar uma repetição desse tipo para calcular e escrever os senos de todos os ângulos integrais entre 0 e $n$, inclusive:

**Exemplo 3.7**

```
Algoritmo calcula_senos

  variável
    n, i: inteiro

  leia(n)
  i ← 0
  enquanto i ≥ n faça
    escreva(seno(i))
```

```
    i ← i + 1
  fim enquanto
fim
```

Nesse exemplo é importante observar que a variável `i` precisa receber um valor inicial (igual a zero) antes da cláusula `enquanto` para garantir que a repetição vai ser executada da forma planejada. Para ilustrar uma condição um pouco mais complexa, podemos alterar o exemplo anterior para calcular apenas o seno de ângulos com menos de 45 graus, enquanto forem menores que $n + 1$ (que é equivalente a serem menores ou iguais a $n$).

**Exemplo 3.8**

```
Algoritmo calcula_senos

  variável
    n, i: inteiro

  leia(n)
  i ← 0
  enquanto (i < 45) e (i < n + 1) faça
    escreva(seno(i))
    i ← i + 1
  fim enquanto
fim
```

A condição que controla a repetição pode ser qualquer expressão válida com valor lógico. Evidentemente, para que a repetição pare 'algo' deve acontecer dentro do laço que mude o resultado da condição. No caso do exemplo, o incremento do valor de `i` (com prévia inicialização) garante que em determinado momento a condição será atingida.

Conforme apresentado na Seção 2.2.8, uma expressão lógica conectada pelo operador **e** é verdadeira apenas se ambos os termos da expressão são verdadeiros. Assim, a repetição do exemplo anterior pára quando qualquer uma das expressões for falsa, ou seja, quando `i` atingir 45 ou quando `i` atingir `n + 1`, o que acontecer primeiro. Como exemplo, verifica-se que, se $n = 35$ o algoritmo escreve os senos dos ângulos 0, 1, 2, ..., 35. Se $n = 60$ o algoritmo escreve os senos dos ângulos 0,1,2, ..., 44.

Pode-se ter uma estrutura que se repete para sempre, por exemplo:

**Exemplo 3.9**

```
enquanto verdadeiro faça
  comandos
fim enquanto
```

Essa repetição é chamada de **loop infinito**. Um loop infinito pode acontecer também quando cometemos algum erro ao especificar a condição lógica que controla a repetição ou ao esquecer de algum comando dentro da iteração, como no exemplo abaixo:

**Exemplo 3.10**

```
Algoritmo calcula_senos

  variável
    n, i: inteiro

  leia(n)
  i = 0
  enquanto (i < 45) e (i < n + 1) faça
    escreva(seno(i))
  fim enquanto
fim
```

Nesse exemplo, a variável `i` não é incrementada como deveria e o algoritmo escreve o seno de zero indefinidamente.

Não obstante, loops infinitos podem aparecer ocasionalmente em um programa correto que funcione continuamente.

Em muitos casos um procedimento que recebe dados do usuário e executa uma tarefa deve ser repetido várias vezes, até que o usuário decida parar o processo. Como exemplo admite-se que o algoritmo da compra de passes de ônibus deva executar não para uma, mas para várias compras. É preciso determinar uma condição de continuidade do algoritmo (ou reversamente, uma condição de parada). Será considerado que uma categoria de passageiro de valor zero indique fim de processamento, e os demais valores dêem continuidade ao processo. O algoritmo, nesse caso, ficaria:

**Exemplo 3.11**

```
Algoritmo ônibus

  constante
    N_CATEGORIAS = 6
    IDOSO = 1
```

```
  ESTUDANTE = 2
  FAIXA_I = 3
  FAIXA_II = 4
  ESTUDANTE_E_FAIXA_I = 5


tipo
  categorias = 0 até N_CATEGORIAS

variável
  passagem_integral: real
  categoria: categorias
  desconto: inteiro
  valor_final: real
  número_de_passes: inteiro



leia(passagem_integral)
leia(categoria)
enquanto categoria ≠ 0 faça
leia(número_de_passes)
  selecione categoria entre
  IDOSO:
    desconto ← 100
  ESTUDANTE:
    desconto ← 50
  FAIXA_I:
    desconto ← 50
  FAIXA_II:
    desconto ← 25
  ESTUDANTE_E_FAIXA_I:
    desconto ← 75
  senão
    desconto ← 0
  fim seleção
  valor_final ← número_de_passes *
              (passagem_integral - desconto/100 * passagem_integral)
  escreva(valor_final)
  leia(categoria)
fim enquanto
```

```
fim
```

No algoritmo acima, pode-se observar que, uma vez que a variável `categoria`, que é um dado de entrada, é utilizada no teste de continuidade (`categoria` $\neq$ `0`), ela deve ser lida antes da primeira entrada no laço de execução do `enquanto`, e lida novamente antes do final da repetição, para permitir o processamento do próximo valor da variável `categoria` ou a parada da repetição. Observe sempre esse fato quando utilizando repetições com condição controlada por dados lidos do usuário.

Um outro uso importante de estruturas de repetição é a generalização de determinados procedimentos. Como exemplo, admite-se que vai ser criado um algoritmo para calcular a soma de vários valores numéricos lidos do teclado. Se esse número de valores da soma for pré-definido e pequeno (3 ou 4 números, por exemplo), o programa é bastante simples. Pode-se ter um variável para cada número lido e somar essas variáeis, conforme o algoritmo abaixo (supondo 4 variáveis a serem somadas):

```
Algoritmo somasimples

  variável
    valor1, valor2, valor2, valor4: real
    soma: real

  leia(valor1, valor2, valor3, valor4)
  soma ← (valor1 + valor2 + valor3 + valor4)/4
  escreva(soma)
fim
```

Evidentemente, o algoritmo acima é muito simples, e de pouca utilidade por si só. Porém, dentro de um contexto maior, em um algoritmo com mais tarefas, é até provável.

No entanto, pode-se desejar criar um algoritmo que não deixe fixa a quantidade de números na soma, ou seja, que varie a quantidade de números somados a cada execução. Uma forma de fazer isso é ler e somar, em seqüencia, os vários números digitados pelo usuário, até que seja digitado um valor que indique final da entrada de dados. Ele pode ser um número fora do intervalo esperado para uma parcela da soma. No próximo exemplo, admite-se que -100 é esse valor. A repetição então lê um número, acumula seu valor à soma dos números lidos previamente e lê o próximo número usando a mesma variável de leitura, até que seja digitado o sinal de parada, isto é, um valor

menor que -100. Este último, evidentemente, não deve fazer parte da soma. O algoritmo (tente fazer antes de olhar), é dado no próximo exemplo.

**Exemplo 3.12** *Soma consecutiva de valores digitados pelo usuário*

```
Algoritmo soma

  variável
    número, soma: real

  soma ← 0
  leia(número)
  enquanto número > -100 faça
    soma ← soma + número
    leia(número)
  fim enquanto
  escreva(soma)
fim
```

No algoritmo acima, a variável `soma` é acrescida do valor da variável `número` a cada vez. A primeira leitura da variável `número` deve ser feita antes da primeira passagem pelo teste da cláusula `enquanto`, e novamente no final da repetição, para permitir o funcionamento correto da estrutura de repetição. Além disso, a inicialização da variável `soma`, também chamada de **acumuladora**, é de extrema importância, para garantir a correta realização da soma consecutiva. No exemplo, a digitação pelo usuário do valor -100 (ou qualquer outro menor que ele) pára o processo.

**Exercício Sugerido 7** *Desenvolva o algoritmo do produto consecutivo de uma seqüencia de números. Dica: a inicialização de uma variável acumuladora é sempre o elemento neutro da operação que ela deve acumular.*

Não fica difícil, neste ponto, alterar o algoritmo do Exemplo 3.12 para que ele calcule a média aritmética dos números lidos, ao invés de apenas a sua soma. A média aritmética é a soma das parcelas, dividida pelo número delas. Para realizar esse cálculo, um item necessário é o número de parcelas, que a atual forma do algoritmo não fornece. Isso também não é sabido antecipadamente, portanto precisa ser calculado. Para isso, precisamos de uma variável contadora, que acumule a quantidade de valores lidos. O algoritmo resultante é apresentado a seguir.

**Exemplo 3.13**

```
Algoritmo média

  variável
    número, soma: real
    n: inteiro


  soma ← 0
  n ← 0
  leia(número)
  enquanto número > -100 faça
    soma ← soma + número
    n ← n + 1
    leia(número)
  fim enquanto
  média ← soma/n
  escreva(soma, n, média)
fim
```

**Exercício Sugerido 8** *Desenvolver um algoritmo para calcular a soma da série:*

$$S = 1 + \frac{1}{x} + \frac{1}{x^2} + \frac{1}{x^3} + \ldots + \frac{1}{x^n},$$

*n lido do usuário.*

## 3.4 Outra forma de repetição por condição

Uma outra forma de repetição é aquela que possui o teste no final do bloco de comandos. Uma diferença dessa forma para a anterior é que os comandos dentro da estrutura são executados uma vez antes que a condição seja testada pela primeira vez, e serve para os processos iterativos onde existe garantia de execução correta do bloco pelo menos uma vez. Outra diferença com relação à forma anterior de repetição é que enquanto aquela estabelecia uma condição de continuidade, esta estabelece a condição de parada para a repetição. O seu formato é dado por:

```
repita
  comandos
até condição
```

Nesse caso, os comandos são repetidos uma ou mais vezes, até que a condição se torne verdadeira (isto é, enquanto a condição for falsa).

Como exemplo admita usar uma repetição desse tipo para ler pelo menos um número do teclado, calcular e imprimir seu quadrado e parar depois de imprimir o primeiro zero:

**Exemplo 3.14**

```
Algoritmo calcula_quadrado

  variável
    número: real

  repita
    leia(número)
    escreva(número*número)
  até número = 0
fim
```

Também neste tipo de repetição a condição lógica pode envolver várias variáveis, constantes, funções e operadores.

A escolha de uma repetição com teste no início ou de uma com teste no fim ocorre quando os comandos serão executados pelo menos zero vezes ou pelo menos uma vez, respectivamente.

Como um exemplo adicional de uso desse tipo de repetição, admite-se um algoritmo que deva ler um conjunto de números digitados pelo usuário, e calcular a sua soma até que ela supere o valor 25500 ou até que tenham sido lidos 150 números (qualquer dessas condições deve parar a repetição). Quando a repetição terminar, o algoritmo deve imprimir o quanto a soma excede o valor limite (se for o caso) e o número de termos da soma. O algoritmo é apresentado a seguir.

**Exemplo 3.15**

```
Algoritmo calcula_soma

  constante
    LIMITANTE = 25500
    NÚMERO_DE_TERMOS = 150

  variável
    número: inteiro
```

```
    soma: inteiro
    cont: inteiro

  soma ← 0
  cont ← 0
  repita
    leia(número)
    soma ← soma + número
    cont ← cont + 1
  até (soma > LIMITANTE) ou (cont = NÚMERO_DE_TERMOS)
  escreva (cont,soma)
  se soma > LIMITANTE então
    escreva (limitante - soma)
  fim se
fim

fim
```

Em muitas circunstâncias o número de vezes que um bloco de comandos vai executar é definido antes do início da repetição. Existe uma estrutura em pseudo-código para expressar precisamente essa situação, a qual é apresentada a seguir.

## 3.5 Repetição por contagem

Na iteração baseada em contagem, sabe-se antecipadamente quantas vezes um conjunto de comandos vai ser repetido. O formato da repetição por contatem é dado a seguir:

```
para variável de valor_inicial até valor_final passo
valor_do_passo faça
  comandos
fim para
```

Inicialmente a `variável`, que chamamos de variável controladora, é inicializada com o `valor_inicial`. Essa variável e o valor devem ser do tipo inteiro. Em seguida, os comandos são repetidos zero ou mais vezes, enquanto o valor da variável estiver entre o `valor_inicial` e o `valor_final`, inclusive. No final de cada repetição do conjunto de comandos, a variável controladora é automaticamente acrescida do `valor_do_passo` e o teste do limite é repetido.

Por exemplo, consideremos o algoritmo que conta de um até `n`, n digitado pelo usuário:

**Exemplo 3.16**

```
Algoritmo conta_com_para

  variável
    n, i: inteiro

  leia(n)
  para i de 1 até n passo 1 faça
    escreva(i)
  fim para
fim
```

No algoritmo acima, na primeira vez que o programa 'passa' pela cláusula `para`, o valor 1 é atribuído à variável `i`, e comparado com o valor final (no caso, o valor da variável `n`). Se o valor de `i` é menor ou igual a `n` o comando de escrita dentro da estrutura de repetição é executado. Após a escrita, `i` é acrescido do valor do passo, nesse caso 1, e o processo comparação → comandos → incremento de `i` se repete até que o valor de `i` ultrapasse o valor de `n`.

**Exercício Sugerido 9** *Escreva o resultado do algoritmo acima para os valores de* `n`*: 10, 1, e -1.*

Uma repetição desse tipo pode não executar nenhuma vez seu bloco de comandos, como no exemplo abaixo em que o valor inicial já é maior que o valor final:

**Exemplo 3.17**

```
Algoritmo não_conta_com_para

  variável
    i: inteiro

  para i de 11 até 10 passo 1 faça
    escreva(i)
  fim para
fim
```

A estrutura de controle `para` admite uma variação em que a repetição se dá em ordem decrescente. Para isso, basta que o valor do passo seja negativo. Por exemplo, consideremos o algoritmo que conta de `n` até um, `n` digitado pelo usuário:

**Exemplo 3.18**

```
Algoritmo conta_decrescente

  variável
    i,n: inteiro

  leia (n)
  para i de n até 1 passo -1 faça
    escreva(i)
  fim para
fim
```

Da mesma forma que na versão crescente, o bloco de comandos não será executado se o valor-inicial da variável for menor que o valor-final.

Além das formas crescente e decrescente, podemos alterar o valor do passo para permitir acréscimo (ou decréscimo) da variável controladora do laço por mais de 1 a cada vez. Os algoritmos abaixo imprimem em tela a contagem crescente de 1 até um valor lido, de 2 em 2, e a contagem decrescente de três em três a partir de um valor lido até atingir 1.

**Exemplo 3.19**

```
Algoritmo conta_crescente_de_2

  variável
    i: inteiro
    n: inteiro

  leia(n)
  para i de 1 até n passo 2 faça
    escreva(i)
  fim para
fim
```

Como resultado do exemplo acima são impressos os valores $1, 3, 5, ..., t, t = n$ se `n` ímpar ou $t = n - 1$ se `n` par.

**Exemplo 3.20**

```
Algoritmo conta_decrescente_de_3

  variável
    i: inteiro

  leia(n)
  para i de n até 1 passo -3 faça
    escreva(i)
  fim para
fim
```

Como resultado do exemplo acima são impressos os valores $n, n-3, n-6, ..., t, t = 1, 2$ ou 3 dependendo do valor de `n`.

Com base nessa estrutura, pode-se desenvolver com mais naturalidade algoritmos para os quais já se saiba antecipadamente quantas vezes um trecho de comandos deve repetir. Esse é o caso, por exemplo, da exponenciação, que é a multiplicação de um número por ele mesmo um número fixo de vezes. Com o uso dessa estrutura de repetição, o algoritmo do Exemplo 3.6 poderia ser generalizado da seguinte forma:

**Exemplo 3.21**

```
Algoritmo expoente

  variável
    base, expoente, resultado: inteiro
    i: inteiro

  leia(base,expoente)
  resultado ← 1
  Para i de 1 até expoente passo 1 faça
    resultado ← resultado*base
  fim para
  escreva (resultado)
fim
```

O exemplo acima funciona bem, mesmo para os valores de expoente zero e um (conferir, percorrendo o algoritmo para os valores zero e um, e para outros valores). No entanto, a implementação de um algoritmo como esse deve levar em conta o fato de que o valor da variável `resultado` cresce rapidamente, e alguma providência para prevenção do valor excessivo da variável pode ser necessária, dependendo do uso real do algoritmo.

**Evitar a todo custo 1** *A variável controladora do laço de contagem pode ser utilizada dentro dos comandos, como demonstrado nos exemplos anteriores, que imprimiram seu valor. No entanto, é de extrema importância em laços controlados por contagem não alterar a variável controladora, deixando que a própria estrutura gerencie seu valor. Não se deve, por exemplo, 'forçar' o final de um laço atribuindo à variável um valor fora dos limites da repetição. Um laço que necessite dessa manipulação deve ser substituído por uma outra forma de repetição. Como exemplo, é dado o trecho de algoritmo abaixo:*

```
Algoritmo laço_ruim

  constante
    LIMITANTE = 25500

  variável
    n, soma, i: inteiro

  leia (n)
  soma ← 0
  para i de 1 até n passo 1 faça
    leia(termo)
    soma ← soma + termo
    se soma > LIMITANTE então
      i ← n + 1
    fim se
  fim para
  escreva (soma)
fim
```

*O algoritmo acima teve a intenção de parar o laço antes da próxima leitura (isto é, antes do n-ésimo termo), caso a soma ultrapasse um limitante.*

*Isso é uma prática ruim de programação, que torna o entendimento e acompanhamento do código muito difícil. O laço acima pode ser facilmente substituído pela estrutura abaixo:*

```
Algoritmo laço_bom

  constante
    LIMITANTE = 25500
```

```
  variável
    n, soma, i: inteiro

  leia (n)
  soma ← 0
  i ← 1
  Enquanto (i ≤ n) e (soma ≤ LIMITANTE) faça
    leia(termo)
    soma ← soma + termo
    i ← i + 1
  fim enquanto
  escreva (soma)
fim
```

*que é muito mais inteligível.*

No texto abaixo são resolvidos dois exercícios, que o leitor deve tentar fazer por si antes de olhar a resposta.

**Exercício Resolvido 3** *Cálculo de Fatorial.*

**Enunciado** *Desenvolver um algoritmo para calcular o fatorial de um número inteiro, definido como:*

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

***Resolução 3*** *Dado um número inteiro positivo, um algoritmo para calcular o seu fatorial deve executar a expressão acima, isto é, um produto consecutivo de vários fatores. A cada multiplicação, o fator é o fator da multiplicação anterior menos 1.*

*Uma forma de implementar isso é usar um laço controlado por contagem e, a cada repetição, usar a variável controladora como fator no produto. Essa solução é dada a seguir.*

```
Algoritmo fatorial

  variável
    n, termo: inteiro
    produto: inteiro

    leia(n)
    produto ← 1
```

```
    para termo de n até 2 passo -1 faça
      produto ← produto*termo
    fim para
    escreva (produto)
fim
```

*No caso do algoritmo acima, o valor final da variável controladora é 2 pois não há a necessidade de multiplicar pelo último fator, que é 1, já que ele não altera o resultado.*

**Exercício Resolvido 4** *Cálculo de exp(x).*
*É possível calcular o valor de $e^x$ através da série:*

$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$

**Enunciado***: Desenvolver um algoritmo para calcular o valor da série acima de modo que o mesmo difira do valor calculado através da função pré-definida* `exp` *de, no máximo, 0,0001. O programa deverá escrever o valor de* `x`*, o valor calculado através da série, o valor dado pela função exp e o número de termos utilizados da série, para 20 valores de* `x` *digitados pelo usuário.*

***Resolução 4*** *A solução do problema através de pseudo-código demandará o uso de, no mínimo, duas estruturas de repetição aninhadas: uma delas para controlar a soma consecutiva da série e outra, mais externa, para executar tudo 20 vezes, conforme solicitado no enunciado. Assim, o esqueleto do algoritmo poderia ser dado da seguinte forma:*

```
para i de 1 até 20 faça
  leia(x)
  série ← 1
  n_termos ← 1
  enquanto abs(serie-exp(x)) > 0,0001 faça
    calcule o numerador
    calcule o denominador
    série ← série + numerador/denominador
    n_termos ← ntermos + 1
  fim do enquanto
  imprima os dados
fim para
```

*No algoritmo acima, observa-se o cuidado com alguns itens, como a inicialização da série já com o primeiro termo (1), a utilização do valor absoluto*

*da diferença entre o valor calculado e o valor da função, a acumulação do valor da série e a contagem de termos.*

*É necessário continuar o detalhamento do algoritmo. Pensando no passo* `calcule o numerador`*, pode-se observar que o numerador varia de um termo para outro pela multiplicação do numerador anterior pelo valor de* `x`*. Portanto, para determinar o numerador basta inicializar uma variável com o primeiro e, a cada repetição, multiplicá-lo por* `x`*. O algoritmo abaixo faz essa adição ao algoritmo.*

```
para i de 1 até 20 faça
  leia(x)
  série ← 1
  numerador ← 1
  n_termos ← 1
  enquanto abs(serie-exp(x)) > 0,0001 faça
    numerador ← numerador * x
    calcule o denominador
    série ← série + numerador/denominador
    n_termos ← ntermos + 1
  fim do enquanto
  imprima os dados
fim para
```

*Pensando agora no passo* `calcule o denominador`*, nota-se pela formulação da série que ele envolve o cálculo de um fatorial. A cada passo, o número do qual se calcula o fatorial é acrescido de um ao número do termo anterior. Utilizando o algoritmo do fatorial apresentado no exercício resolvido 3, a adição desse passo resulta no algoritmo abaixo.*

```
para i de 1 até 20 faça
  leia(x)
  série ← 1
  fat_num ← 1
  numerador ← 1
  n_termos ← 1
  enquanto abs(serie-exp(x)) > 0,0001 faça
    numerador ← numerador * x
    {Calcule o denominador}
    denominador ← 1
    para termo de fat_num até 2 passo -1 faça
      denominador ← denominador*termo
```

```
    fim para
    série ← série + numerador/denominador
    fat_num ← fat_num + 1
    n_termos ← ntermos + 1
  fim do enquanto
  imprima os dados
fim para
```

*Este é o momento na confecção do algoritmo de completar as definições (variáveis, constantes, etc.), mas principalmente de revisar a sua estrutura.*

*O algoritmo está correto, e, a menos de exceções que ainda não foram tratadas, o procedimento é simples: calcula-se o numerador, o denominador, e a soma. No entanto, observa-se que o fatorial poderia ser calculado incrementalmente cada passo, pois o fatorial de um termo é o fatorial do termo anterior multiplicado pelo inteiro seguinte. Por exemplo,* $5!$ *é* $4! \times 5$*. Portanto, o algoritmo está executando muitas operações de multiplicação desnecessárias. Assim, a próxima versão do algoritmo, embora menos clara que o algoritmo anterior em termos de correspondência com a definição numérica da série, é também correta e mais eficiente.*

*Também pode-se notar que o valor de* `exp(x)` *na versão anterior do algoritmo é calculado a cada passagem pelo teste da cláusula* `enquanto`*. Isso é desnecessário, uma vez que, para um dado valor de* `x`*,* `exp(x)` *é fixo. Ele pode ser calculado uma única vez, antes de entrar no laço, como na versão final abaixo.*

```
Algoritmo série_exp

  constante
    N = 20
    DIF = 0,0001

  variável
    i, n_termos, fator, denominador: inteiro
    x, série, numerador, exp_x: real



  para i de 1 até N faça
    leia(x)
    denominador ← 1
    série ← 1
```

```
    numerador ← 1
    fator ← 1
    n_termos ← 1
    exp_x ← exp(x)
    enquanto abs(serie-exp_x) > DIF faça
      numerador ← numerador * x
      denominador ← denominador * fator
      fator ← fator + 1
      série ← série + numerador/denominador
      n_termos ← n_termos + 1
    fim do enquanto
    imprima (x,exp_x,serie,n_termos)
  fim para
fim
```

*O aspecto de eficiência deve ser avaliado pelo programador e decisões devem ser tomadas quanto à utilização de uma ou outra forma de expressar um algoritmo ou solucionar um problema.*

*Além das modificações mencionadas acima, foram adicionadas ao algoritmo definições de constantes.*

## 3.6 Equivalência entre as três formas de repetição

É possível perceber que apenas uma forma de iteração (por exemplo, aquela da cláusula `enquanto`), seria suficiente para desenvolver qualquer algoritmo baseado em repetição. As demais formas existem para facilitar a estruturação dos algoritmos e aumentar a clareza do código.

Por exemplo, qualquer laço de repetição baseado em contagem crescente é equivalente a uma estrutura `enquanto` no seguinte formato:

```
variável ← valor_inicial

enquanto variável ≤ valor_final faça
  comandos
  variável ← variável + valor_do_passo
fim enquanto
```

Como ilustração, o exemplo abaixo mostra uma outra versão do algoritmo apresentado no Exemplo 3.6, usando agora a estrutura `enquanto`.

**Exemplo 3.22**

```
Algoritmo expoente

  variável
    base, expoente, resultado: integer;
    i: integer;

  leia(base,expoente)
  resultado ← 1
  i ← 1
  Enquanto i ≤ expoente faça
    resultado ← resultado*base
    i ← i + 1
  fim para
  escreva (resultado)
fim
```

No texto abaixo é apresentada uma alternativa, usando a estrutura `para`, ao algoritmo apresentado no Exemplo 3.7.

**Exemplo 3.23**

```
Algoritmo calcula_senos

  variável
    n, i: real

  leia(n)
  Para i de 0 até i passo n faça
    escreva(seno(i))
  fim enquanto
fim
```

## 3.7 Percorrendo um Algoritmo: Casos de Teste

Já foi mencionado que toda vez que um algoritmo é desenvolvido ele deve ser revisado buscando melhorias. Além disso, é preciso verificar se sua execução

está correta. Um recurso para iniciar esse processo é percorrer o algoritmo. Isso demanda simular manualmente a execução de cada passo do algoritmo até chegar ao fim, assumindo valores para aqueles dados que são lidos do usuário. A seguir é dado um exemplo disso.

**Exemplo 3.24** *Percorrendo um algoritmo.*

*Suponha o algoritmo para descobrir o maior valor entre três valores digitados pelo usuário, apresentado no Exemplo 3.3. Suponha também que, para ser testado, ele vai ser repetido 3 vezes, num laço de repetição, conforme o algoritmo abaixo:*

```
1  Algoritmo maiordetres
2
3    variável
4      maior: inteiro
5      número1, número2, número3: inteiro
6      i: inteiro
7
8
9    Para i de 1 até 3 passo 1 faça
10     leia(número1, número2, número3)
11     se número1>número2 então
12       se (número2>número3) ou (número2 = número3) então
13         maior ← número1
14       senão
15         se número1>número3 então
16           maior ← número1
17         senão
18           maior ← número3
19         fim se
20       fim se
21     senão
22       se (número2 > número3) ou (número2 = número3) então
23         maior ← número2
24       senão
25         maior ← número3
26       fim se
27     fim se
28     escreva(maior)
29   fim para
30 fim
```

*O comando da linha 10 introduz os valores nas variáveis. Supondo que, nesse ponto, o usuário digitou os valores 3 1 2 na primeira vez, 1 4 4 na segunda, e 5 4 1 na terceira. A tabela a seguir pode ser preenchida com uma linha a cada passo, de acordo com os passos executados do algoritmo a partir desses dados.*

| *Passo* | *Linha* | *i* | *numero1* | *numero2* | *número3* | *maior* | *condição* |
|---|---|---|---|---|---|---|---|
| *1* | *9* | *1* | *?* | *?* | *?* | *?* | $1 \leq 3$ |
| *2* | *10* | *1* | *3* | *1* | *2* | *?* | |
| *3* | *11* | *1* | *3* | *1* | *2* | *?* | *verdadeiro* |
| *4* | *12* | *1* | *3* | *1* | *2* | *?* | *falso* |
| *5* | *13* | *1* | *3* | *1* | *2* | *3* | |
| *6* | *28* | *1* | *3* | *1* | *2* | *3* | |
| *7* | *29* | *2* | *3* | *1* | *2* | *3* | |
| *8* | *9* | *2* | *3* | *1* | *2* | *3* | $2 \leq 3$ |
| *9* | *10* | *2* | *1* | *4* | *4* | *3* | |
| *10* | *11* | *2* | *1* | *4* | *4* | *3* | *falso* |
| *11* | *22* | *2* | *1* | *4* | *4* | *3* | *verdadeiro* |
| *12* | *23* | *2* | *1* | *4* | *4* | *4* | |
| *13* | *28* | *2* | *1* | *4* | *4* | *4* | |
| *14* | *29* | *3* | *1* | *4* | *4* | *4* | |
| *15* | *9* | *3* | *1* | *4* | *4* | *4* | $3 \leq 3$ |
| *16* | *10* | *3* | *5* | *4* | *1* | *4* | |
| *17* | *11* | *...* | | | | | |

Na tabela do exemplo acima, que ficou incompleta para o último caso (valores lidos 5, 4 e 1), pode-se perceber a evolução do algoritmo passo a passo. Esse é um exercício necessário para verificar o funcionamento da solução dada ao problema antes de realizar a implementação.

**Exercício Sugerido 10** *Completar a tabela acima até que o algoritmo termine, ou seja, a variável controladora* `i` *ultrapasse o valor 3.*

Percorrer um algoritmo algumas vezes é a única forma de garantir sua estrutura antes da implementação. Isso ocorre principalmente para os problemas mais complexos, e os algoritmos com muitos testes e repetições, para os quais não se consegue provar a correção.

Para conferir a execução de um algoritmo, é preciso estabelecer um conjunto de dados de entrada que permita explorar os vários caminhos de decisão e as várias possibilidades de repetição. Por exemplo, para se testar adequadamente um algoritmo para encontrar o maior entre três números, é preciso ter um conjunto de dados de teste que explore todas as possíveis relações de

ordem entre os três números, bem como a possibilidade de existirem números iguais.

Um conjunto de casos de teste ideal explora todas as vias de decisão de um algoritmo. No entanto, muitas vezes não é possível estabelecer um conjunto de dados de entrada que atenda a esses requisitos. Quando isso acontece, é preciso que o conjunto de casos efetivamente utilizado compreenda uma boa amostra do conjunto completo, e que procure definir o comportamento do algoritmo em casos extremos.

Para citar alguns exemplos: no algoritmo do Exercício Resolvido 1 os casos de testes devem apresentar casos extremos de nota, com e sem nota substitutiva, e casos que cubram possibilidades dos dois tipos de média (acima e abaixo de 5,0 para provas e trabalhos); no caso do cálculo dos senos (Exemplo 3.8) é preciso testar vários valores inclusive os limítrofes (0 e 44 graus), e acima de 45 graus; no caso do cálculo das passagens de ônibus (Exemplo 3.11), todas as categorias devem ser testadas para os dois tipos de cartela.

**Exercício Sugerido 11** *Definir casos de teste para todos os algoritmos exemplificados neste capítulo e para todos os exercícios sugeridos. Percorrê-los.*