INOVAÇÃO
E TECNOLOGIA

unidade
2

Algorítimos e PROGRAMAÇÃO

UNIVERSIDADE LaSalle EAD

Ferramenta VisuAlg, Comandos Condicionais, Teste de Mesa

Prezado(a) estudante

Estamos começando uma unidade desta disciplina. Os textos que a compõem foram organizados com cuidado e atenção, para que você tenha contato com um conteúdo completo e atualizado tanto quanto possível. Leia com dedicação, realize as atividades e tire suas dúvidas com os tutores. Dessa forma, você com certeza alcançará os objetivos propostos para essa disciplina.

OBJETIVO GERAL

Conceituar, identificar, exemplificar e codificar com a utilização da ferramenta VisuAlg, ordenar dados, realizar comandos condicionais simples, compostos e de múltiplas escolha.

OBJETIVOS ESPECÍFICOS

- Analisar algoritmos sequenciais em forma de pseudocódigo desenvolvidos na ferramenta VisuAlg.
- Demonstrar algoritmos sequenciais em pseudocódigo na ferramenta VisuAlg.
- Usar o pseudocódigo na representação da solução de problemas.
- Analisar algoritmos com estruturas condicionais simples em fluxograma.
- Identificar problemas que necessitam de comandos condicionais simples para a sua solução.
- Resolver problemas através da aplicação de comandos condicionais simples em fluxograma.
- Analisar algoritmos com estruturas condicionais simples em pseudocódigo.
- Identificar problemas que necessitam de comandos condicionais simples para a sua solução.
- Resolver problemas através da aplicação de comandos condicionais simples em pseudocódigo.
- Analisar algoritmos com comandos condicionais duplos ou compostos em fluxograma.

OBJETIVOS ESPECÍFICOS

- Desenvolver algoritmos em fluxograma que precisam, para a sua solução, de comandos condicionais duplos ou compostos.
- Compreender a estrutura básica de um comando de seleção duplo ou composto em fluxograma.
- Analisar algoritmos com estruturas condicionais compostas e encadeadas em pseudocódigo.
- Identificar problemas que necessitam de comandos condicionais compostos para a solução.
- Desenvolver algoritmos em pseudocódigo que necessitam de comandos condicionais compostos e encadeados para a solução.
- Analisar algoritmos com estruturas de seleção múltipla em fluxograma.
- Identificar problemas que podem ser resolvidos com a aplicação de comando de seleção múltipla.
- Construir algoritmos em fluxograma que necessitam de comandos de seleção múltipla para a solução.
- Analisar algoritmos com estruturas de seleção múltipla em pseudocódigo
- Identificar problemas que podem ser resolvidos com a aplicação de comando de seleção múltipla.
- Desenvolver algoritmos em pseudocódigo que necessitam de comandos de seleção múltipla para a sua solução.
- Reconhecer a importância de um teste de mesa.
- Usar o teste de mesa em algoritmos na forma de pseudocódigo e fluxograma.
- Contrastar os tipos de erros de sintaxe e semântica.
- Diferenciar o funcionamento dos métodos de pesquisa sequencial e binário.
- Analisar algoritmos utilizando os métodos de pesquisa binário e sequencial.
- Usar a pesquisa de dados na solução de problemas em pseudocódigo.
- Diferenciar alguns dos métodos de ordenação interna de dados.
- Reconhecer o funcionamento de alguns dos métodos de ordenação interna de dados.
- Identificar os benefícios da aplicação da ordenação de dados na solução de problemas do dia a dia.

Parte 1

Desenvolvimento de Algoritmos Sequenciais Através de Pseudocódigo (Ferramenta VisuAlg)

3.4 → fluxograma de programas sequenciais

No Capítulo 1 foram mostrados alguns blocos utilizados em fluxogramas, incluindo os que representam comandos de entrada, de saída e de atribuição (Figura 1.4). Diversas formas para representar entradas e saídas podem ser encontradas na literatura. Nos blocos adotados nesse livro é utilizado o mesmo bloco para ambas, identificando claramente a ação a ser executada (entrada ou saída). A Figura 3.1 mostra um fluxograma em que é feita uma entrada de dados que preenche a variável `valor`, sendo, em seguida, informado qual o valor lido.

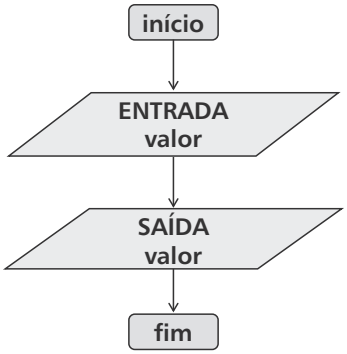


figura 3.1 Fluxograma com entrada e saída de dados.

Um programa pode ter vários comandos de entrada e de saída de dados em lugares diferentes. As formas dos blocos que representam esses comandos mostram visualmente, no fluxograma, os pontos de interação do programa com o usuário.

O comando de atribuição é representado através de um retângulo, dentro do qual é escrito o nome da variável, o símbolo que representa a atribuição (←) e a expressão, em sua forma matemática, ou seja, sem necessidade de representá-la em uma só linha. Como exemplo, a Figura 3.2 mostra o fluxograma que corresponde ao problema apresentado na Seção 1.2:

1. `obter os dois valores`
2. `realizar a soma`
3. `informar o resultado`

O primeiro passo corresponde a um comando de entrada de dados, em que são lidos dois valores. Para armazenar os valores lidos devem ser declaradas duas variáveis, `valor1` e `valor2`, de tipos compatíveis com os valores que serão fornecidos na entrada. No segundo passo, é

realizada a operação de soma dos valores contidos nas duas variáveis, sendo o resultado armazenado em outra variável chamada de `soma`. O terceiro passo corresponde a um comando de saída, através do qual o valor armazenado na variável `soma` é informado ao usuário. As setas indicam a sequência em que os comandos são executados.

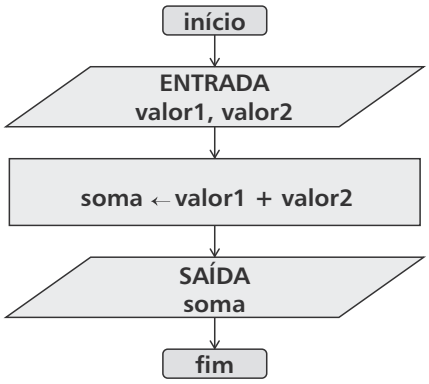


figura 3.2 Fluxograma da soma de dois valores.

3.5 estrutura de um algoritmo

Nesta seção será montado o primeiro algoritmo completo utilizando as declarações e os comandos vistos até aqui. Será utilizado o mesmo exemplo da seção anterior (soma de dois valores), para o qual já foi construído o fluxograma.

Um algoritmo deve sempre iniciar com um **cabeçalho**, no qual o objetivo do algoritmo deve ser claramente identificado. A primeira linha desse cabeçalho deve trazer o nome do algoritmo, o qual, por si só, deve dar uma indicação das ações a serem executadas pelo mesmo. No caso do exemplo, o algoritmo foi chamado de `Soma2`, pois vai efetuar a soma de dois valores. Na linha seguinte do cabeçalho, na forma de um comentário, deve ser explicado o objetivo do algoritmo. Essa explicação é útil principalmente nos casos em que o nome do algoritmo não é suficientemente autoexplicativo. Cabeçalho do exemplo utilizado:

```
Algoritmo Soma2
{INFORMA A SOMA DE 2 VALORES LIDOS}
```

Logo após o cabeçalho vem a seção das **declarações** de variáveis, de constantes e de tipos. Para facilitar o entendimento de um algoritmo, é importante identificar claramente as `variáveis de entrada` e `de saída`, pois elas fazem a interface do usuário com o programa. As demais variáveis utilizadas durante o processamento, denominadas `variáveis auxiliares`, são declaradas em uma linha especial. Essa separação desaparece ao se traduzir o algoritmo para uma linguagem de programação, mas é aconselhável que seja acrescentada ao programa na forma de um comentário.

A declaração de variáveis do `Algoritmo Soma2` é a seguinte:

```
Entradas: valor1, valor2 (real)    {VALORES LIDOS}
Saídas: soma (real)
```

Os nomes escolhidos para as variáveis devem ser curtos e indicar qual a informação que elas irão armazenar. Caso isso não fique claro somente através do nome escolhido, é aconselhável escrever comentários explicando o significado de cada variável.

Após a seção de declarações, vem a área de **comandos**, delimitada pelas palavras reservadas `início` e `fim`. Cada comando deve ser escrito em uma linha separada. Ao contrário das linguagens de programação Pascal e C, a pseudolinguagem utilizada não emprega símbolo para separar comandos, sendo essa separação identificada somente pela posição de cada comando no algoritmo.

É importante utilizar **comentários** ao longo do algoritmo, indicando as ações que estão sendo executadas em cada passo. Isso auxilia muito os testes e a depuração do programa.

A estrutura básica de um algoritmo, com os elementos discutidos até o momento, é:

```
Algoritmo <nome do algoritmo>
{descrição do objetivo do algoritmo}
<declarações>
início
<comandos>
fim
```

Em declarações aparecem com frequência alguns ou todos os seguintes elementos:

```
Entradas: <lista de nomes de variáveis com seus tipos>
Saídas: <lista de nomes de variáveis com seus tipos>
Variáveis auxiliares: <lista de nomes de variáveis com seus tipos>
```

O algoritmo completo do exemplo da soma de dois valores é:

```
Algoritmo 3.1 - Soma2
{INFORMA A SOMA DE DOIS VALORES LIDOS}
  Entradas: valor1, valor2 (real){VALORES LIDOS}
  Saídas:   soma (real)
início
  ler (valor1, valor2)           {OBTÉM OS 2 VALORES}
  soma ← valor1 + valor2         {CALCULA A SOMA}
  escrever (soma)                {INFORMA A SOMA}
fim
```

Nos exercícios de fixação a seguir, recomenda-se definir inicialmente o(s) resultado(s) a produzir, a(s) entrada(s) a obter e, só então, tentar determinar um modo de solução. Procurar

identificar, nas soluções fornecidas, quais as linhas que correspondem, respectivamente, à entrada de dados, ao processamento e à apresentação dos resultados.

Observar que todos os problemas discutidos seguem o esquema básico destacado no início deste capítulo: entrada de dados, processamento e saída de dados.

3.6 exercícios de fixação

exercício 3.1 Fazer um programa que recebe três notas de alunos e fornece, como saídas, as três notas lidas, sua soma e a média aritmética entre elas.

A Figura 3.3 mostra o fluxograma deste programa. Inicialmente são lidas as três notas, que são também impressas para que o usuário possa verificar o que foi lido. Em seguida, é calculada e informada a soma. Finalmente, é efetuado o cálculo da média, que é também informado ao usuário. A utilização de diversos comandos de saída neste programa permite ao programador verificar quais os valores intermediários do processamento, auxiliando a depurar o programa.

O algoritmo desse programa acrescenta as declarações das variáveis utilizadas, que não aparecem no fluxograma. São incluídos também comentários para explicar os diferentes passos do algoritmo.

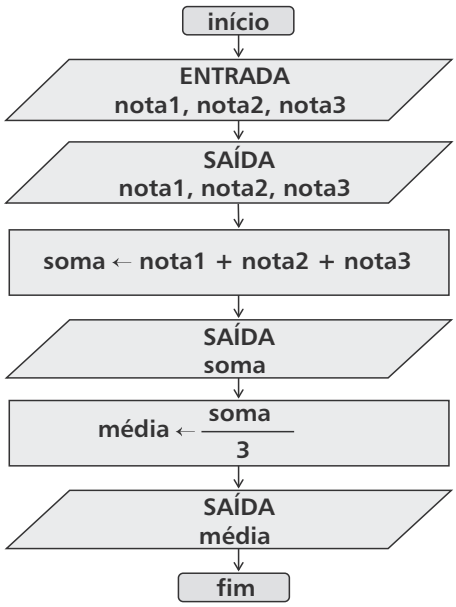


figura 3.3 Fluxograma do cálculo da média de três notas.

```
Algoritmo Média1
{INFORMA A SOMA E A MÉDIA DAS 3 NOTAS DE UM ALUNO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: soma, média (real)
início
  ler (nota1, nota2, nota3)            {ENTRADA DAS 3 NOTAS}
  escrever (nota1, nota2, nota3)       {INFORMA AS NOTAS LIDAS}
  soma ← nota1 + nota2 + nota3         {CALCULA A SOMA}
  escrever (soma)                      {INFORMA SOMA}
  média ← soma / 3                     {CALCULA A MÉDIA}
  escrever (média)                     {INFORMA MÉDIA CALCULADA}
fim
```

exercício 3.2 Dado o raio de um círculo, construir um algoritmo que calcule e informe seu perímetro e sua área.

Fórmulas: perímetro = $2 \times \pi \times$ raio
área = $\pi \times$ raio^2

```
Algoritmo Círculo
{INFORMA O PERÍMETRO DE UM CÍRCULO E SUA ÁREA}
  Entrada: raio (real)
  Saídas: perímetro, área (real)
início
  ler (raio)                           {LÊ O RAIO DO CÍRCULO}
  perímetro ← 2 * 3,14 * raio          {CALCULA O PERÍMETRO}
  área ← 3,14 * sqr(raio)              {CALCULA A ÁREA}
  escrever (perímetro, área)           {INFORMA PERÍMETRO E ÁREA}
fim
```

exercício 3.3 Dado o preço de um produto em reais, converter este valor para o equivalente em dólares. O programa deverá ler do teclado o preço do produto e a taxa de conversão para o dólar.

```
Algoritmo ConversãoParaDólar
{CONVERTE UM VALOR EM REAIS PARA DÓLARES}
  Entradas: preço (real)    {PREÇO EM REAIS}
            taxa (real)     {TAXA DE CONVERSÃO PARA DÓLAR}
  Saída: emdolar (real)     {PREÇO EM DÓLARES}
início
  ler (preço)               {LÊ O PREÇO EM REAIS}
  ler (taxa)                {LÊ A TAXA DO DÓLAR}
  emdolar ← preço * taxa    {CALCULA O VALOR EM DÓLARES}
  escrever (emdolar)        {INFORMA O VALOR EM DÓLARES}
fim
```

exercício 3.4 Escrever um algoritmo que calcula a comissão de um vendedor sobre uma venda efetuada. O algoritmo deve ler o número de um vendedor, o valor da venda efetuada e o percentual a receber sobre a venda.

```
Algoritmo ComissãoSobreVenda
{CALCULA A COMISSÃO DE UM VENDEDOR SOBRE UMA VENDA}
  Entradas: numVendedor (inteiro)      {NÚMERO DO VENDEDOR}
            valorVenda (real)          {VALOR DA VENDA}
            percentual (real)          {PERCENTUAL A RECEBER}
  Saídas: comissão (real)              {COMISSÃO A RECEBER}
início
  ler (numVendedor, valorVenda)              {LEITURA DADOS VENDA}
  ler (percentual)                           {LEITURA PERCENTUAL}
  comissão ← valorVenda * percentual * 0,01 {CÁLCULO DA COMISSÃO}
  escrever (numVendedor, comissão)           {SAÍDA DADOS}
fim
```

exercício 3.5 Permutar o conteúdo de duas variáveis na memória. O programa deverá iniciar preenchendo as duas variáveis por leitura e imprimindo os valores contidos nas variáveis. Em seguida, deve permutar o conteúdo das duas variáveis, ou seja, o conteúdo da primeira deve ficar na segunda e vice-versa. Imprimir novamente as duas variáveis para conferir se seus conteúdos foram realmente trocados.

Esta aplicação é muito comum e deve ser efetuada com cuidado. Ao solicitar a troca dos valores de duas variáveis na memória (a e b), pode-se pensar em fazer o seguinte:

```
a ← b
b ← a
```

Entretanto, ao colocar em a o valor contido em b, o valor que estava em a é perdido, conforme mostra a Figura 3.4. Para que isso não aconteça, o valor em a deve ser previamente guardado em uma variável auxiliar, para depois ser buscado para preencher a variável b, conforme ilustra a Figura 3.5. No algoritmo apresentado a seguir, as duas variáveis são preenchidas por leitura. Os valores nelas contidos são informados antes e depois da troca, para que se possa verificar se a operação foi realizada com sucesso.

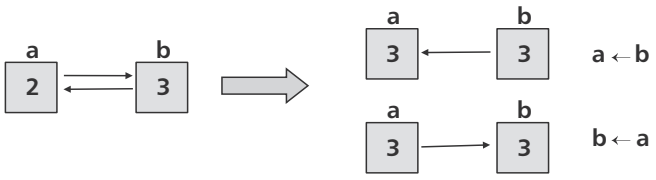


figura 3.4 Troca errada dos conteúdos de duas variáveis.

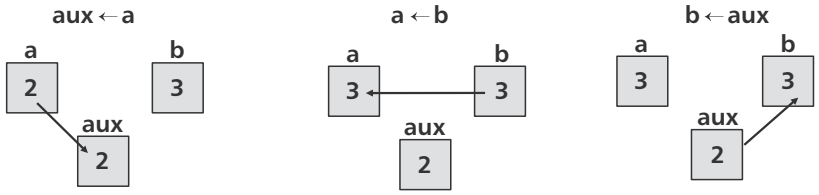


figura 3.5 Troca correta dos conteúdos de duas variáveis.

```
Algoritmo Permuta2Variáveis
{PERMUTA O CONTEÚDO DE DUAS VARIÁVEIS}
  Entradas: a, b (real)          {VARIÁVEIS A SEREM PERMUTADAS}
  Saídas: a, b                   {AS MESMAS VARIÁVEIS}
  Variável auxiliar: aux (real)
início
  ler (a, b)              {LÊ OS DOS DOIS VALORES}
  escrever (a, b)         {INFORMA OS VALORES LIDOS}
  aux ← a                 {PERMUTA O CONTEÚDO DAS DUAS VARIÁVEIS}
  a ← b
  b ← aux
  escrever (a, b)         {INFORMA VALORES TROCADOS}
fim
```

3.7 em Pascal

3.7.1 entrada de dados

Pascal possui dois comandos diferentes para a entrada de dados:

```
read ( <lista de variáveis, separadas por vírgulas> )
readln [( <lista de variáveis, separadas por vírgulas> )]
```

A lista de variáveis será preenchida com os dados na ordem em que forem submetidos. A principal diferença entre os comandos de leitura `read` e `readln` se dá quando são fornecidos mais dados do que os necessários para preencher a lista. No caso do `read`, os dados não usados são guardados em um *buffer* e utilizados no próximo comando de leitura. Se o comando utilizado for `readln`, os dados em excesso são descartados e novos dados serão lidos em uma próxima leitura.

Observe que, no caso do comando `readln`, a lista de variáveis é opcional. Um comando `readln` sem a lista de variáveis faz o programa ficar parado esperando um `enter` do teclado. Ele pode ser utilizado para manter os dados exibidos na tela, permitindo ver os resultados durante o tempo que se quiser. Ao receber o `enter`, a execução do programa continua e, caso ela termine, não se verá mais o que foi exibido. Ao longo deste livro, além da formatação

Parte 2

Comandos Condicionais Simples (Fluxograma)

Os algoritmos que solucionam os problemas apresentados até o momento são puramente sequenciais: todas as instruções são executadas na ordem em que foram definidas, uma após a outra, sem exceção. Este capítulo introduz uma nova classe de problemas, na qual uma ou mais ações podem ou não ser executadas, dependendo da avaliação prévia de condições. Para resolver esses problemas são apresentados três novos comandos que possibilitam a alteração do fluxo sequencial de execução de um programa.

No capítulo anterior, foi resolvido o problema do cálculo da média aritmética de três notas de um aluno em uma disciplina. Estendendo essa aplicação, o professor que utiliza esse programa quer que, além da média das notas obtidas na disciplina, seja informado se o aluno foi aprovado (no caso de sua média ser igual ou superior a 6). Essa informação não pode ser obtida somente com o conjunto de comandos do Capítulo 3, uma vez que requer a análise de uma condição e a execução de uma ação somente se a condição analisada for verdadeira.

Outra possibilidade é fornecer não apenas a informação de aprovação, mas também a de reprovação. Nesse caso, também condicionada ao conteúdo da média calculada, ocorre a execução de apenas uma de duas ações mutuamente exclusivas: se a média for maior ou igual a 6, o programa informa que o aluno foi aprovado; caso contrário, se a média for inferior a 6, informa que o aluno foi reprovado.

Seguindo adiante nesse raciocínio, pode-se escrever um trecho de programa em que, dependendo da média obtida, também é informado o conceito correspondente à média calculada. Aqui, diferentes faixas de valor de uma mesma informação desencadeiam ações diferentes, também mutuamente exclusivas, isto é, a execução de uma ou mais instruções específicas está associada ao valor da informação.

Os comandos que solucionam esses problemas são apresentados neste capítulo.

4.1 comando de seleção simples

Um **comando de seleção simples**, também chamado de comando condicional, permite que a execução de um trecho do programa dependa do fato de uma condição ser verdadeira, isto é, vincula a execução de um ou mais comandos ao resultado obtido na avaliação de uma expressão lógica (também denominada expressão condicional). O comando de seleção simples é sempre composto por uma condição e um comando. A condição é expressa por uma expressão lógica, cuja avaliação produz um resultado `verdadeiro` ou `falso`. A sintaxe de um comando de seleção simples é:

```
se <expressão lógica>
então <comando>
```

Observe que o comando somente é executado se o resultado da expressão lógica for `verdadeiro`; se o resultado for `falso`, nada é executado.

Por meio do comando de seleção simples pode-se, por exemplo, condicionar a exibição da informação de que um aluno foi aprovado somente para o caso de sua média ser igual ou superior a 6:

```
se média ≥ 6
então escrever('Aprovado')
```

A execução do comando `escrever` ocorre apenas quando a condição for verdadeira, ou seja, quando o conteúdo da média for igual ou superior a 6. Nada é executado se a média for inferior a 6.

A Figura 4.1 representa o fluxograma de um comando de seleção simples. Um novo tipo de bloco, com formato de losango, é utilizado para representar a realização de um teste, escrevendo-se dentro desse bloco a expressão lógica a ser avaliada. Esse bloco tem duas saídas, uma para o caso da expressão ser avaliada como verdadeira, e outra para quando o resultado da avaliação da expressão for falso. As informações que correspondem a cada saída devem estar claramente identificadas. O fluxograma mostra com clareza que nada é executado no caso do resultado da avaliação da expressão ser falso.

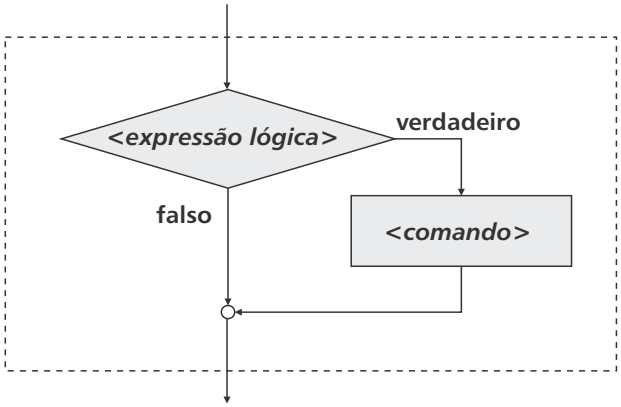


figura 4.1 Fluxograma de um comando de seleção simples.

O algoritmo a seguir informa, além da média de um aluno, se ele foi aprovado:

```
Algoritmo 4.1 – Média2
{INFORMA A MÉDIA DAS 3 NOTAS DE UM ALUNO E SE ELE FOI APROVADO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          {Informação de aprovado}
início
  ler (nota1, nota2, nota3)      {ENTRADA DAS 3 NOTAS}
  média ←(nota1 + nota2 + nota3)/3
  escrever (média)               {INFORMA MÉDIA CALCULADA}
  se média ≥ 6
  então escrever('Aprovado')     {INFORMA SE ALUNO FOI APROVADO}
fim
```

A Figura 4.2 apresenta o fluxograma relativo a esse algoritmo.

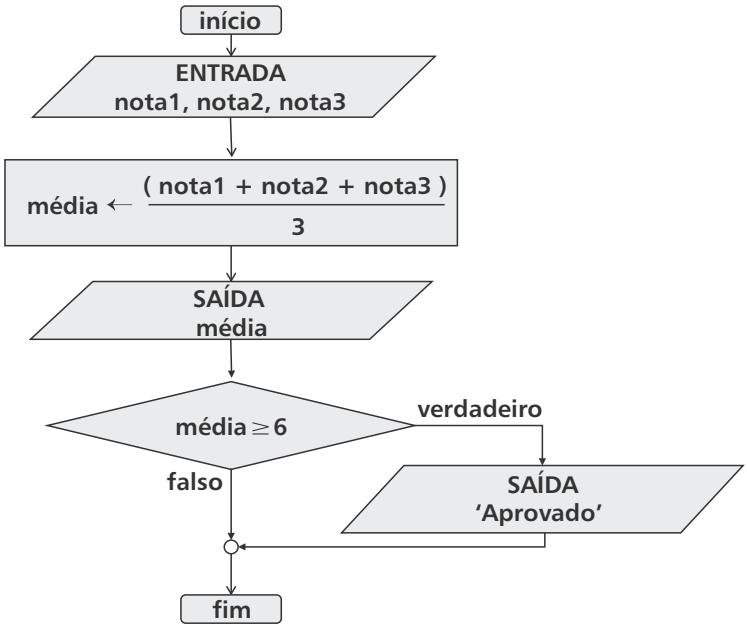


figura 4.2 Fluxograma de um exemplo com comando de seleção simples.

4.2 comando composto

Na sintaxe do comando de seleção simples nota-se que somente um comando pode ser executado caso a condição seja verdadeira. Mas o que fazer quando se quer executar vários comandos condicionados à avaliação de uma mesma expressão lógica? Por exemplo, supondo que, na aplicação anterior, se queira saber a média somente no caso das três notas lidas serem iguais ou superiores a 6, então a média deve ser calculada e informada somente se a condição for verdadeira.

Para que isso seja possível, é necessário que dois comandos, o de cálculo da média e o de saída dessa média, sejam executados quando a condição for verdadeira. Como a sintaxe do comando de seleção simples exige a execução de um único comando, faz-se necessária a definição de um novo tipo de comando, denominado **comando composto**.

Na pseudolinguagem aqui utilizada, um comando composto é delimitado pelas palavras reservadas `início` e `fim`. Sintaticamente, trata-se de um único comando. Quaisquer comandos podem ser incluídos dentro de um comando composto. Algumas linguagens de programação permitem, inclusive, a definição de novas variáveis dentro de um comando composto, as quais são alocadas na memória apenas no momento em que inicia a execução desse coman-

do e são liberadas no término da execução do comando. Contudo, essa possibilidade não será considerada neste livro.

O problema proposto no início desta seção pode ser resolvido com a utilização desse novo recurso, conforme mostrado no Algoritmo `Média3`:

```
Algoritmo 4.2 – Média3
{INFORMA MÉDIA DO ALUNO SOMENTE SE SUAS 3 NOTAS FOREM IGUAIS OU SUPE-
RIORES A 6}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          {Informação de aprovado}
início
  ler (nota1, nota2, nota3)                          {ENTRADA DAS 3 NOTAS}
  se (nota1 ≥ 6) e (nota2 ≥ 6) e (nota3 ≥ 6)
  então início                                       {COMANDO COMPOSTO}
        média ← (nota1 + nota2 + nota3) / 3          {CALCULA MÉDIA}
        escrever (média)                             {INFORMA MÉDIA}
        fim
fim
```

O fluxograma desse algoritmo, apresentado na Figura 4.3, mostra com clareza que o conjunto de comandos pode não ser executado, dependendo do resultado da condição.

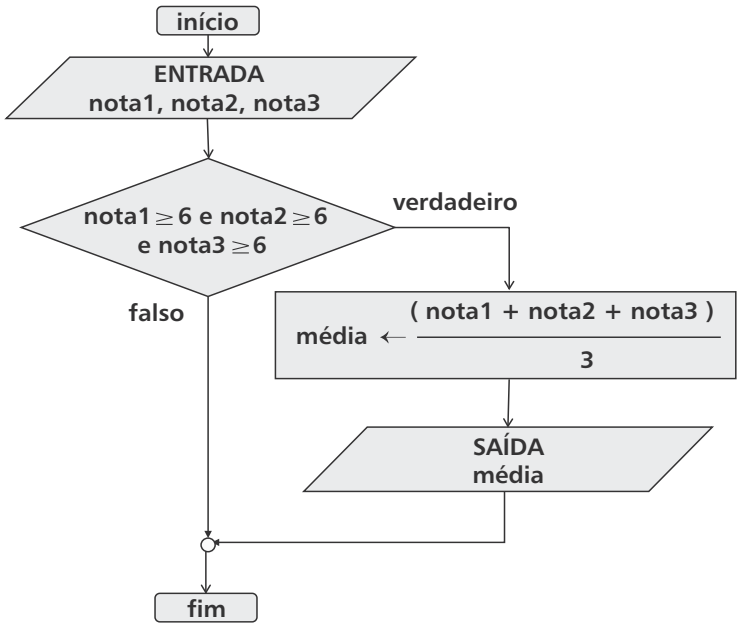


figura 4.3 Fluxograma de comando de seleção simples com comando composto.

ANOTAÇÕES

Parte 3

Comandos Condicionais Simples (Pseudocódigo)

Os algoritmos que solucionam os problemas apresentados até o momento são puramente sequenciais: todas as instruções são executadas na ordem em que foram definidas, uma após a outra, sem exceção. Este capítulo introduz uma nova classe de problemas, na qual uma ou mais ações podem ou não ser executadas, dependendo da avaliação prévia de condições. Para resolver esses problemas são apresentados três novos comandos que possibilitam a alteração do fluxo sequencial de execução de um programa.

No capítulo anterior, foi resolvido o problema do cálculo da média aritmética de três notas de um aluno em uma disciplina. Estendendo essa aplicação, o professor que utiliza esse programa quer que, além da média das notas obtidas na disciplina, seja informado se o aluno foi aprovado (no caso de sua média ser igual ou superior a 6). Essa informação não pode ser obtida somente com o conjunto de comandos do Capítulo 3, uma vez que requer a análise de uma condição e a execução de uma ação somente se a condição analisada for verdadeira.

Outra possibilidade é fornecer não apenas a informação de aprovação, mas também a de reprovação. Nesse caso, também condicionada ao conteúdo da média calculada, ocorre a execução de apenas uma de duas ações mutuamente exclusivas: se a média for maior ou igual a 6, o programa informa que o aluno foi aprovado; caso contrário, se a média for inferior a 6, informa que o aluno foi reprovado.

Seguindo adiante nesse raciocínio, pode-se escrever um trecho de programa em que, dependendo da média obtida, também é informado o conceito correspondente à média calculada. Aqui, diferentes faixas de valor de uma mesma informação desencadeiam ações diferentes, também mutuamente exclusivas, isto é, a execução de uma ou mais instruções específicas está associada ao valor da informação.

Os comandos que solucionam esses problemas são apresentados neste capítulo.

4.1 comando de seleção simples

Um **comando de seleção simples**, também chamado de comando condicional, permite que a execução de um trecho do programa dependa do fato de uma condição ser verdadeira, isto é, vincula a execução de um ou mais comandos ao resultado obtido na avaliação de uma expressão lógica (também denominada expressão condicional). O comando de seleção simples é sempre composto por uma condição e um comando. A condição é expressa por uma expressão lógica, cuja avaliação produz um resultado `verdadeiro` ou `falso`. A sintaxe de um comando de seleção simples é:

```
se <expressão lógica>
então <comando>
```

Observe que o comando somente é executado se o resultado da expressão lógica for `verdadeiro`; se o resultado for `falso`, nada é executado.

Por meio do comando de seleção simples pode-se, por exemplo, condicionar a exibição da informação de que um aluno foi aprovado somente para o caso de sua média ser igual ou superior a 6:

```
se média ≥ 6
então escrever('Aprovado')
```

A execução do comando `escrever` ocorre apenas quando a condição for verdadeira, ou seja, quando o conteúdo da média for igual ou superior a 6. Nada é executado se a média for inferior a 6.

A Figura 4.1 representa o fluxograma de um comando de seleção simples. Um novo tipo de bloco, com formato de losango, é utilizado para representar a realização de um teste, escrevendo-se dentro desse bloco a expressão lógica a ser avaliada. Esse bloco tem duas saídas, uma para o caso da expressão ser avaliada como verdadeira, e outra para quando o resultado da avaliação da expressão for falso. As informações que correspondem a cada saída devem estar claramente identificadas. O fluxograma mostra com clareza que nada é executado no caso do resultado da avaliação da expressão ser falso.

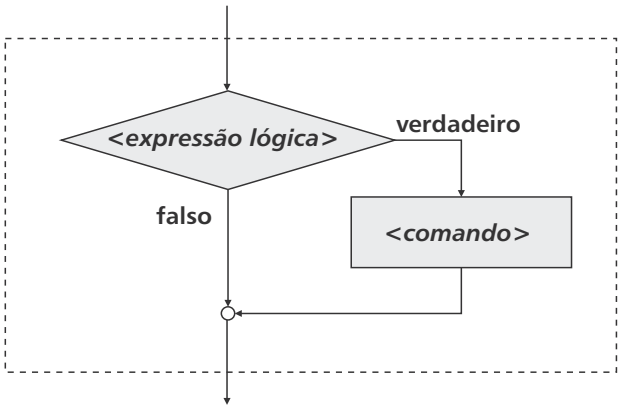


figura 4.1 Fluxograma de um comando de seleção simples.

O algoritmo a seguir informa, além da média de um aluno, se ele foi aprovado:

```
Algoritmo 4.1 – Média2
{INFORMA A MÉDIA DAS 3 NOTAS DE UM ALUNO E SE ELE FOI APROVADO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          {Informação de aprovado}
início
  ler (nota1, nota2, nota3)      {ENTRADA DAS 3 NOTAS}
  média ←(nota1 + nota2 + nota3)/3
  escrever (média)               {INFORMA MÉDIA CALCULADA}
  se média ≥ 6
  então escrever('Aprovado')     {INFORMA SE ALUNO FOI APROVADO}
fim
```

A Figura 4.2 apresenta o fluxograma relativo a esse algoritmo.

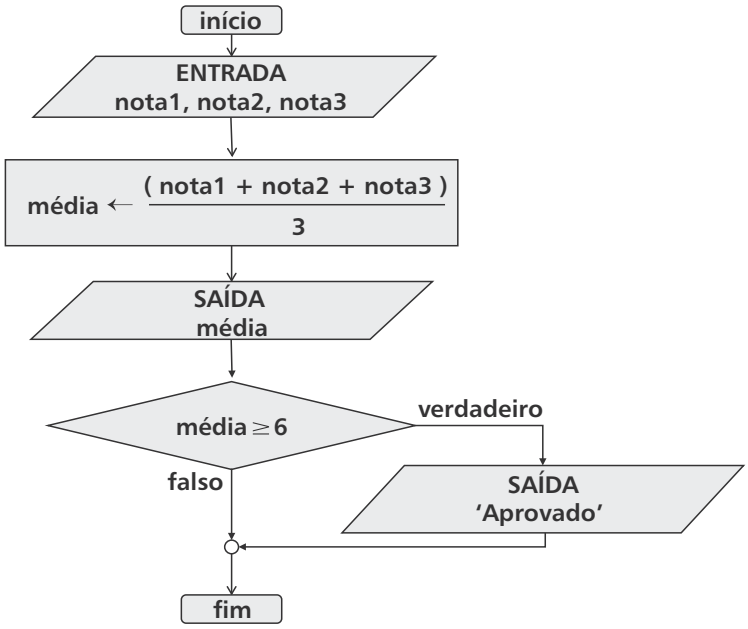


figura 4.2 Fluxograma de um exemplo com comando de seleção simples.

4.2 comando composto

Na sintaxe do comando de seleção simples nota-se que somente um comando pode ser executado caso a condição seja verdadeira. Mas o que fazer quando se quer executar vários comandos condicionados à avaliação de uma mesma expressão lógica? Por exemplo, supondo que, na aplicação anterior, se queira saber a média somente no caso das três notas lidas serem iguais ou superiores a 6, então a média deve ser calculada e informada somente se a condição for verdadeira.

Para que isso seja possível, é necessário que dois comandos, o de cálculo da média e o de saída dessa média, sejam executados quando a condição for verdadeira. Como a sintaxe do comando de seleção simples exige a execução de um único comando, faz-se necessária a definição de um novo tipo de comando, denominado **comando composto**.

Na pseudolinguagem aqui utilizada, um comando composto é delimitado pelas palavras reservadas `início` e `fim`. Sintaticamente, trata-se de um único comando. Quaisquer comandos podem ser incluídos dentro de um comando composto. Algumas linguagens de programação permitem, inclusive, a definição de novas variáveis dentro de um comando composto, as quais são alocadas na memória apenas no momento em que inicia a execução desse coman-

do e são liberadas no término da execução do comando. Contudo, essa possibilidade não será considerada neste livro.

O problema proposto no início desta seção pode ser resolvido com a utilização desse novo recurso, conforme mostrado no Algoritmo `Média3`:

```
Algoritmo 4.2 – Média3
{INFORMA MÉDIA DO ALUNO SOMENTE SE SUAS 3 NOTAS FOREM IGUAIS OU SUPE-
RIORES A 6}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          {Informação de aprovado}
início
  ler (nota1, nota2, nota3)                          {ENTRADA DAS 3 NOTAS}
  se (nota1 ≥ 6) e (nota2 ≥ 6) e (nota3 ≥ 6)
  então início                                       {COMANDO COMPOSTO}
        média ← (nota1 + nota2 + nota3) / 3          {CALCULA MÉDIA}
        escrever (média)                             {INFORMA MÉDIA}
        fim
fim
```

O fluxograma desse algoritmo, apresentado na Figura 4.3, mostra com clareza que o conjunto de comandos pode não ser executado, dependendo do resultado da condição.

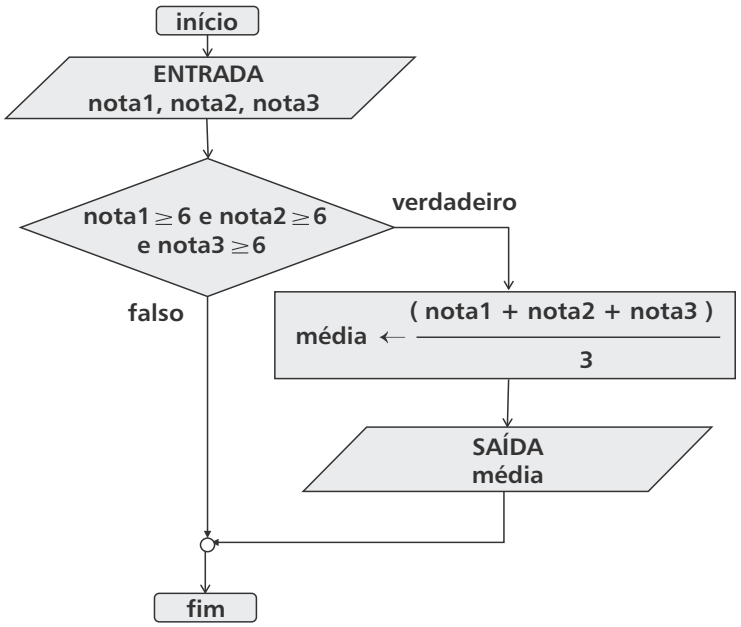


figura 4.3 Fluxograma de comando de seleção simples com comando composto.

Parte 4

Teste de Mesa

1.6 dicas

Critérios que devem ser observados ao construir um algoritmo:

- procurar soluções simples para proporcionar clareza e facilidade de entendimento do algoritmo;
- construir o algoritmo através de refinamentos sucessivos;
- seguir todas as etapas necessárias para a construção de um algoritmo de qualidade;
- identificar o algoritmo, definindo sempre um nome para ele no cabeçalho. Este nome deve traduzir, de forma concisa, seu objetivo. Por exemplo: `Algoritmo 1.1 – Soma2` indica, através do nome, que será feita a soma de dois valores;
- definir, também no cabeçalho, o objetivo do algoritmo, suas entradas e suas saídas;
- nunca utilizar desvios incondicionais, como `GOTO` (`VÁ PARA`).

1.7 testes

Testes de mesa. É importante efetuar, sempre que possível, testes de mesa para verificar a eficácia (corretude) de um algoritmo antes de implementá-lo em uma linguagem de programação. Nestes testes, deve-se utilizar diferentes conjuntos de dados de entrada, procurando usar dados que cubram a maior quantidade possível de situações que poderão ocorrer durante a utilização do algoritmo. Quando o algoritmo deve funcionar apenas para um intervalo definido de valores, é recomendável que se simule a execução para valores válidos, valores limítrofes válidos e inválidos e valores inválidos acima e abaixo do limite estabelecido. Por exemplo, se um determinado algoritmo deve funcionar para valores inteiros, no intervalo de 1 a 10, inclusive, o teste de mesa deveria incluir a simulação da execução para, no mínimo, os valores 0, 1, 10, 11 e um valor negativo.

3.10 testes

incluir comandos de saída para depurar programas. Uma forma de depurar um programa é usar diversos comandos de saída ao longo do programa para acompanhar os valores que são assumidos por algumas das variáveis durante o processamento. Uma vez feitos todos os testes necessários, esses comandos devem ser retirados do programa.

Por exemplo, no Algoritmo 3.1 poderia ser acrescentado um comando de saída logo após a leitura, para verificar se os valores lidos são mesmo aqueles que foram fornecidos. Para facilitar a remoção desses comandos auxiliares do programa, sugere-se que sejam alinhados de forma diferente dos demais comandos:

```
Algoritmo 3.1 – Soma2
  {INFORMA A SOMA DE DOIS VALORES LIDOS}
  Entradas: valor1, valor2 (real){VALORES LIDOS}
  Saídas:   soma (real)
início
  ler (valor1, valor2)              {ENTRADA DOS 2 VALORES}
  escrever (valor1, valor2)         {PARA TESTE}
  soma ← valor1 + valor2            {CALCULA A SOMA}
  escrever (soma)                   {INFORMA A SOMA}
fim
```

testar para todos os dados de entrada possíveis. Outro aspecto importante para garantir resultados corretos é realizar testes com todas as possíveis combinações de dados de entrada, incluindo valores positivos, negativos e nulos. Testar também o que acontece quando são fornecidos tipos de dados incorretos na entrada. Na Tabela 3.7, são mostrados alguns pares de valores de entrada que poderiam ser utilizados para testar o Algoritmo 3.1 (adaptando os números à sua representação na linguagem de programação utilizada):

tabela 3.7 Exemplos de valores de entrada a serem testados

valor1	valor2	
0	0	dois valores nulos
0	5	primeiro valor nulo
-2	0	segundo valor nulo
-3	0	um valor negativo e um nulo
0	2.3	um valor real e um nulo
5	5	dois valores iguais
2	5	dois valores inteiros diferentes
3.5	2.7	dois valores reais diferentes
100000	-3.7	um positivo e um negativo
-7	-4.67	dois valores negativos

3.11 exercícios sugeridos

exercício 3.1 Escreva uma expressão lógica que seja verdadeira no caso do valor contido em uma variável inteira `valor` estar compreendido entre os valores 10 e 50, incluindo os limites.

exercício 3.2 Leia as coordenadas de dois pontos no plano cartesiano e imprima a distância entre esses dois pontos. Fórmula da distância entre dois pontos ($x1$, $y1$) e ($x2$, $y2$):

$$\sqrt{(x2-x1)^2+(y2-y1)^2}$$

exercício 3.3 Dados três valores armazenados nas variáveis a, b e c, calcule e imprima as médias aritmética, geométrica e harmônica desses valores. Calcule também a média ponderada, considerando peso 1 para o primeiro valor, peso 2 para o segundo e peso 3 para o terceiro.

Fórmulas: média aritmética: $\frac{a+b+c}{3}$

média geométrica: $\sqrt[3]{a \times b \times c}$

média harmônica: $\frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$

média ponderada: $\frac{1a+2b+3c}{1+2+3}$

usar comandos aninhados em sequências de testes. Quando vários testes relacionados tiverem que ser feitos, utilizar preferencialmente comandos de seleção dupla aninhados, em vez de sequências de comandos de seleção simples.

usar seleção múltipla em lugar de sequências de seleções simples. Para maior clareza, utilizar um comando de seleção múltipla em vez de sequências de comandos de seleção simples sempre que a linguagem oferecer essa construção.

não repetir desnecessariamente testes semelhantes. Para testes mutuamente exclusivos, utilizar comandos de seleção dupla, evitando assim a repetição desnecessária de testes.

planejar os testes cuidadosamente. Planejar bem os testes para verificar o maior número possível de casos diferentes. Quanto mais testes bem planejados forem realizados, maior a garantia de que o programa será executado corretamente.

4.10 → testes

testar o maior número possível de caminhos de execução. Diversos testes devem ser realizados para testar a corretude de um trecho de programa que contém um comando de seleção. Devem ser criados conjuntos de dados que façam a execução do programa passar pelo maior número de caminhos possíveis. Tanto comandos de seleção simples quanto de seleção dupla devem ser testados com pelo menos dois conjuntos de dados, um que faça a expressão lógica resultar verdadeira e outro que a torne falsa. Por exemplo, o comando:

```
se média ≥ 6
então escrever('Aprovado')
```

deve ser testado com dados que resultem em (1) média maior que 6, (2) média igual a 6 e (3) média menor que 6. Os mesmos conjuntos de dados podem ser utilizados para testar o comando:

```
se média ≥ 6
então escrever('Aprovado')
senão escrever('Reprovado')
```

verificando se as mensagens corretas são apresentadas para os vários conjuntos de dados utilizados.

Um cuidado especial deve ser tomado para criar os conjuntos de dados no caso de comandos aninhados a fim de que o maior número possível de caminhos seja testado. Por exemplo, no comando:

```
se média ≥ 9
então conceito ← 'A'
senão se média ≥ 7,5
      então conceito ← 'B'
      senão se média ≥ 6,0
```

```
então conceito ← 'C'
senão conceito ← 'D'  {MÉDIA < 6}
```

devem ser criados dados para testar cada um dos intervalos de média, incluindo os limites de cada um deles. Por exemplo, testar com valores que resultem nas seguintes médias: 10,0 – 9,5 – 9,0 – 8,0 – 7,5 – 7,0 – 6,0 – 4,0 – 0,0.

testar com dados incorretos. Outro cuidado a ser tomado é fazer testes com valores de dados incorretos para ver como o programa se comporta nesses casos. A execução do programa não deve parar quando forem fornecidos dados incorretos. Quando isso acontecer, o programa deve informar ao usuário a ocorrência de erro. Por exemplo, no código anteriormente citado, caso as notas fornecidas resultem em uma média maior do que 10, o programa atribuirá ao aluno o conceito "A", o que não estará correto. Caso a média resulte em valor menor do que zero, o conceito atribuído será "D", o que também estará incorreto. Se o programa, antes de entrar nesse comando, não fizer um teste da corretude dos dados fornecidos, esse trecho de programa estará sujeito a erro.

testar o pior e o melhor caso. Além dos casos médios, sempre testar o pior e o melhor caso, de acordo com os dados fornecidos.

4.11 exercícios sugeridos

exercício 4.1 Faça um programa que leia dois valores, o primeiro servindo de indicador de operação e o segundo correspondendo ao raio de um círculo. Caso o primeiro valor lido seja igual a 1, calcular e imprimir a área desse círculo. Se o valor lido for 2, calcular e imprimir o perímetro do círculo. Se o valor lido for diferente desses dois valores, imprimir uma mensagem dizendo que foi fornecido um indicador incorreto para a operação a ser realizada.

exercício 4.2 Leia três valores e informe se podem corresponder aos lados de um triângulo. Em caso afirmativo, verifique e informe se esse triângulo é:

a equilátero;
b isósceles;
c escaleno;
d retângulo.

exercício 4.3 Leia três valores e armazene-os nas variáveis A, B e C. Se todos forem positivos, calcule e imprima a área do trapézio que tem A e B por bases e C por altura.

exercício 4.4 Escreva um programa que calcule as seguintes conversões entre sistemas de medida:

a dada uma temperatura na escala Celsius, fornecer a temperatura equivalente em graus Fahrenheit e vice-versa (Fórmula de conversão: $1^{o}\ F = (9 / 5)^{o}\ C + 32$);
b dada uma medida em polegadas, fornecer a equivalente em milímetros e vice-versa (Fórmula de conversão: 1 pol = 24,5 mm).

Parte 5

Comandos de Repetição

5.9 → em C

A linguagem C apresenta peculiaridades significativas nos comandos de repetição. A seguir, será dada ênfase aos aspectos comuns e aos que diferenciam esses comandos dos que foram vistos na linguagem algorítmica e em Pascal.

5.9.1 comando de repetição `for`

A sintaxe do comando de repetição por contagem `for` em C é:

```
for (<expressão 1>; <expressão 2>; <expressão 3>)
    <comando>;
```

A `expressão 1` contém a atribuição do conteúdo inicial a uma ou mais variáveis de controle, as quais são inicializadas antes da primeira execução do comando controlado pelo mecanismo de repetição. No caso de mais de uma inicialização, cada expressão é separada de outra por vírgula. As variáveis de controle devem ser do tipo `int` ou `char`.

A `expressão 2` contém a expressão condicional que, enquanto verdadeira, determina a repetição do comando (simples ou composto) subordinado ao `for`. O uso de mais de uma expressão, separadas por vírgula, é permitido. Nesse caso, o comando do laço será executado apenas quando o resultado lógico de todas as expressões for verdadeiro (um E lógico de todas as expressões).

A `expressão 3` contém a especificação do(s) incremento(s)/decremento(s) que deve(m) ser aplicado(s) à(s) variável(is) de controle após cada execução do comando a ser repetido. Esse(s) incremento(s)/decremento(s) pode(m) ser o resultado de uma expressão aritmética qualquer.

A avaliação da condição do comando `for` ocorre antes do início do trecho a ser repetido. Assim, se essa condição for inicialmente falsa, o comando do laço não será executado.

Em C, a contagem correspondente ao número de repetições não é implementada de forma explícita, uma vez que o encerramento é determinado a partir de uma expressão condicional. Contudo, o número de repetições pode ser determinado a partir da análise dos conteúdos iniciais, incrementos/decrementos aplicados e expressão condicional que determina a repetição. Por isso, é recomendado que as repetições controladas pelo `for` sejam estabelecidas através das expressões e variáveis de controle definidas no cabeçalho do comando.

Em C, é possível inicializar mais de uma variável no início da execução do `for` e alterar mais de uma variável a cada nova iteração. O trecho a seguir gera a tabuada de multiplicação de um valor informado pelo usuário. No `for` que produz a tabuada, a variável `i` é inicializada com 1 e a variável `j`com o valor para o qual deve ser produzida a tabuada. Enquanto `i` for menor ou igual a 10, o trecho é executado. A cada iteração, `i` é incrementado em 1 e `j` é acrescido do valor informado:

```
int i,j,multiplic; // i e' o multiplicando e j e' o produto
printf("\nInforme a tabuada desejada:");
scanf("%d",&multiplic); // entrada do multiplicador desejado
for (i=1, j=multiplic; i <= 10; i++, j=j+ multiplic)
   printf("%3d X %2d = %3d\n", i , multiplic, j);
```

Outra peculiaridade do comando `for` em C é a possibilidade de exclusão de qualquer – ou até todas – as expressões que fazem parte do cabeçalho do comando, mas mantendo sempre todos os separadores. Dessa forma, é sintaticamente correto escrever `for (;;)`, embora essa construção não deva ser utilizada, já que gera um laço infinito.

Na prática, sem comprometer o bom estilo de programação e fazendo uso dessa peculiaridade da implementação do `for` na linguagem C, a inicialização da(s) variável(is) de controle antes do comando `for` é adotada para situações em que limites são obtidos de forma iterativa: neste caso, a `expressão 1` não é definida, mas o ponto e vírgula separador é mantido antes da expressão lógica, como pode ser observado no trecho de programa a seguir:

```
char letra;
printf("Informe letra minuscula inicial: ");
scanf(" %c",&letra);       // inicializa letra por leitura
printf("\n\n");
for (;letra<='z'; letra++) //sem inicializacao, mas com separador;
    printf("%4c", letra);  //imprime letra ate z, usando 4 posicoes
```

A expressão 2 também permite que a quantidade de repetições seja determinada através de expressões mais complexas, mas que respeitam a recomendação de limitar o controle das repetições aos componentes do cabeçalho do comando. Suponha um trecho de programa contendo as instruções necessárias para gerar todos os termos correspondentes ao produto de três números naturais consecutivos que não ultrapasse (o produto gerado) um valor limite especificado. Observar que, nesse caso, a soma dos números corresponde ao termo e também ao limite das repetições, ou seja, termo recebe a soma e soma não deve ultrapassar o limite estabelecido. Em C, a solução desse problema pode ser implementada através do código a seguir:

```
// termo inicial eh 1 X 2 X 3 = 6
for (nro=2; (nro – 1)* nro * (nro + 1) <= limite; nro++)
{
    termo = (nro – 1)* nro * (nro + 1);
    printf("%d ", termo);
}
```

O algoritmo que calcula e informa as médias de 30 alunos é escrito em C como segue:

```
/* Programa Media30
   Informa media dos 30 alunos de uma turma */
#include <stdio.h>  // scanf e printf
#include <stdlib.h> // system
int main()
{
    float nota1, nota2, nota3; // notas informadas
    float media;               // media do aluno – calculada
    int aluno;             // controle do for
    for (aluno=1; aluno <= 30; aluno++) // repete 30 vezes
    {
       printf("\nInforme as 3 notas do aluno %d:", aluno);
       scanf("%f%f%f",&nota1, &nota2, &nota3); //entrada das 3 notas
       media = (nota1 + nota2 + nota3) / 3;    // calculo da media
       printf("\n   Media do aluno %d: %.2f",aluno,media);
    } // fim do for
    system("pause");
    return 0;
}
```

Observar que, como o controle das repetições é efetuado pela expressão condicional, o(s) valor(es) da(s) variável(is) controlada(s) pelo `for`, após o encerramento desse, estará(ão) fora do escopo considerado como válido. Assim, o valor da variável `contador` após a execução do `for` acima é 31.

5.9.2 comando de repetição condicional `while` por avaliação anterior de condição

A sintaxe do comando `while` em C é:

```
while (<condição>)
    <comando>;
```

O comando `while`, quanto à sintaxe, não apresenta diferenças em relação à linguagem algorítmica apresentada, ou seja, o comando, simples ou composto, é repetido enquanto a condição for verdadeira.

O Algoritmo 5.4 é escrito em C da seguinte maneira:

```
/* Programa MediaAlunos_2
   Informa media dos alunos de uma turma. Para indicar fim de
  processamento, o conteúdo informado em nota1 sera' -1.
*/
#include <stdio.h>  // scanf e printf
#include <stdlib.h> // system
int main()
{
    float nota1, nota2, nota3;
    float media;
    int codigo;
    printf("\nInforme as notas do aluno ");
    printf(" (valor negativo na primeira nota para encerrar):");
    scanf("%f %f %f", &nota1, &nota2, &nota3); // as 3 notas
    while (nota1 >= 0) // enquanto nota1 maior ou igual a zero
    {
       printf("\nInforme o codigo do aluno: ");
       scanf("%d",&codigo);    // entrada do codigo do aluno
       media = (nota1 + nota2 + nota3) / 3;       // calculo da media
       printf("\nResultados do aluno %d:\n  Media = %.2f", codigo, media);
       printf("\nInforme as notas do aluno ");
       printf(" (valor negativo na primeira nota para encerrar):");
       scanf("%f %f %f", &nota1, &nota2, &nota3); // as 3 notas
    } // fim do while
    system("pause");
    return 0;
}
```

Usando o comando `while` é possível, em C, construir soluções mais compactas do que em outras linguagens porque, por características da atribuição em C, consegue-se incluir leituras na sua condição.

No trecho a seguir, que utiliza a variável `caract` (caractere), um conjunto de caracteres é lido na condição de um `while` e apresentado na tela. A execução do laço `while` termina quando é fornecido o caractere "#".

```
printf("\nForneca os caracteres desejados, com # ao final: ");
while ((caract=getchar()) != '#') // le caracteres ate encontrar #
      printf("%c", caract);
```

5.9.3 comando de repetição condicional `do/while` por avaliação posterior de condição

Em C, o comando `do/while` corresponde ao comando condicional por avaliação posterior de condição. Isso significa que o comando subordinado, simples ou composto, é executado pelo menos uma vez, independentemente do resultado lógico inicial da expressão de controle. Essa é a única diferença em relação ao comando `while`. Também aqui a repetição ocorre enquanto o valor lógico resultante da avaliação da expressão permanecer verdadeiro.

A sintaxe do comando de repetição `do/while` em C é:

```
do
    <comando>;
while (<expressão condicional >);  // enquanto condicao verdadeira
```

Observar que esse comando não corresponde exatamente ao comando `repita/até` da pseudolinguagem, uma vez que na linguagem C as repetições ocorrem enquanto a condição for verdadeira e, na pseudolinguagem, enquanto for falsa.

No programa a seguir, que efetua a soma de valores inteiros e positivos informados (e consistidos) via teclado, o processo é encerrado a partir da resposta fornecida pelo usuário. Como sempre ocorrerá pelo menos uma leitura de dados, a opção de controle de repetição foi o comando `do/while`. Observar que a digitação do caractere que determina o encerramento da entrada de dados ocorre dentro da própria expressão condicional do comando `do/while` externo.

```
//Soma de um numero indeterminado de valores inteiros positivos
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main (  )
{
  int  soma = 0, valor;
  char resposta;
  do
  {
     do
      {
        printf("\nValor inteiro positivo: ");
```

```
        scanf("%d", &valor);
        if (valor<1)
           printf("\n*** Valor invalido! ***");
      } while (valor<1);
     soma = soma + valor;
     printf("\nDeseja informar outro valor? (S - sim; N - nao): ");
     fflush(stdin);
   } while (toupper(resposta=getchar()) == 'S'); // le em expressao
  printf ("\nSoma = %d\n" , soma);
  system("pause");
  return 0;
}
```

5.9.4 selecionando o comando de repetição mais adequado

O que foi observado na pseudolinguagem sobre como selecionar o comando de repetição mais adequado vale igualmente para a linguagem C. Entretanto, é importante salientar que, em um número elevado de problemas, ao menos dois, quando não os três comandos iterativos vistos, `for`, `while` e `do/while`, podem ser utilizados nas situações em que um laço seja necessário.

Os três trechos a seguir realizam o produto dos mesmos cinco valores. Considera-se que `i` e `prod` (variáveis inteiras) foram inicializados com 1 (um) antes da execução de cada um deles:

```
while (i <= 5) //versao com while
{
  prod = prod * i;
  i++;
}
for (; i <= 5; i++) // versao com for
  prod = prod * i;
do // versao com do/while
 {
  prod = prod * i;
   i++;
 }
  while (i <= 5);
```

5.10 dicas

quando usar comandos `para/faça`. Usar comandos de repetição por contagem (`para/faça`) somente quando o número de repetições for fixo. Quando as repetições dependerem

Parte 6

Comandos Condicionais Compostos (Fluxograma)

4.3 comando de seleção dupla

Supondo que, além de informar a média das três notas do aluno, também se queira que o programa informe se ele foi aprovado (quando a média for igual ou superior a 6) ou reprovado (média inferior a 6). Dois comandos, `se-então`, são necessários para imprimir essas mensagens:

```
se média ≥ 6
então escrever('Aprovado')      {INFORMA SE O ALUNO FOI APROVADO}
se média < 6
então escrever('Reprovado')     {INFORMA SE O ALUNO FOI REPROVADO}
```

Os dois comandos implementam ações mutuamente exclusivas e dependem da avaliação de uma mesma condição, sendo uma das ações associada ao resultado verdadeiro e outra ao resultado falso. Para evitar a repetição da comparação, pode ser utilizado um **comando de seleção dupla** que, a partir do resultado da avaliação de uma condição, seleciona um de dois comandos para ser executado. Sua sintaxe é:

```
se <expressão lógica>
então <comando>
senão <comando>
```

Somente um comando pode ser definido em cada uma das cláusulas `então` e `senão`. Esse comando pode ser simples ou composto, como no caso do comando de seleção simples. O exemplo anterior, resolvido através de dois comandos de seleção simples, equivale ao seguinte comando de seleção dupla:

```
se média ≥ 6
então escrever('Aprovado')
senão escrever('Reprovado')
```

O fluxograma que representa esse comando, mostrado na Figura 4.4, mostra claramente que o fluxo do programa passa por apenas um dos dois comandos, o qual é selecionado pelo resultado da expressão lógica.

O algoritmo que calcula a média de três notas e informa se o aluno foi aprovado ou reprovado é o seguinte:

```
Algoritmo 4.3 – Média4
{INFORMA A MÉDIA DO ALUNO E SE FOI APROVADO OU REPROVADO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          {Informação de aprovado ou reprovado}
início
  ler (nota1, nota2, nota3)     {ENTRADA DAS 3 NOTAS}
  média ← (nota1+nota2+nota3)/3
  escrever (média)              {INFORMA MÉDIA CALCULADA}
```

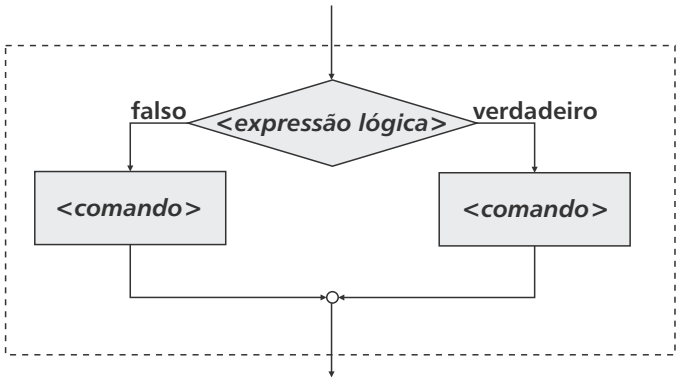


figura 4.4 Fluxograma do comando de seleção dupla.

```
  se média ≥ 6
  então escrever('Aprovado')    {INFORMA SE O ALUNO FOI APROVADO}
  senão escrever('Reprovado')   {INFORMA SE O ALUNO FOI REPROVADO}
fim
```

4.4 → comandos de seleção aninhados

Conforme visto, somente um comando pode ser utilizado nas cláusulas `então` e `senão`, mas não há restrição quanto ao tipo de comando. Pode, inclusive, ser usado um novo comando de seleção – simples ou dupla. Nesse caso, diz-se que os comandos de seleção estão aninhados ou encadeados.

Estendendo um pouco mais a aplicação de apoio a um professor, suponha que se queira obter o conceito do aluno com base na sua média, de acordo com a seguinte conversão:

Conceito A: Média ≥ 9,0
Conceito B: 9,0 > Média ≥ 7,5
Conceito C: 7,5 > Média ≥ 6,0
Conceito D: Média < 6,0

A solução pode ser implementada através de uma sequência de comandos condicionais (opção 1):

```
se média ≥ 9
então conceito ← 'A'
se (média < 9) e (média ≥ 7,5)
então conceito ← 'B'
se (média < 7,5) e (média ≥ 6,0)
então conceito ← 'C'
se (média < 6)
então conceito ← 'D'
```

Nessa sequência de comandos, somente uma das condições será verdadeira e, apesar disso, todas as condições serão sempre avaliadas, desnecessariamente. Para evitar isso, o algoritmo a seguir calcula a média e o conceito, utilizando comandos de seleção dupla aninhados (opção 2). Note que, uma vez encontrada uma condição verdadeira, as que estão após ela, na cláusula senão, não são mais avaliadas. Cabe ressaltar que, nessa solução, não foi feita a análise da validade dos dados de entrada, partindo-se do pressuposto que eles foram corretamente informados.

```
Algoritmo 4.4 - MédiaConceito1
{INFORMA A MÉDIA E O CONCEITO DE UM ALUNO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          conceito (caractere)
início
  ler (nota1, nota2, nota3)       {ENTRADA DAS 3 NOTAS}
  média ← (nota1+nota2+nota3)/3 {CÁLCULO DA MÉDIA}
  escrever (média)                {INFORMA MÉDIA CALCULADA}
  se média ≥ 9                    {CÁLCULO DO CONCEITO}
  então conceito ← 'A'
  senão se média ≥ 7,5
        então conceito ← 'B'
        senão se média ≥ 6,0
              então conceito ← 'C'
              senão conceito ← 'D'  {MÉDIA < 6}
  escrever (conceito)             {INFORMA CONCEITO}
fim
```

Nesse algoritmo, o trecho de programa que calcula o conceito corresponde a um único comando de seleção dupla. Se a média for igual ou superior a 9,0, o conceito "A" é atribuído ao aluno e a execução desse comando termina. No caso dessa condição não ser verdadeira, então é avaliada a segunda condição, que verifica se a média é igual ou superior a 7,5. Se essa condição for verdadeira, o aluno recebe o conceito "B" e o comando é concluído. Se não for verdadeira, então a média é novamente analisada, dessa vez verificando se é maior ou igual a 6,0. Finalmente, independentemente da condição ser verdadeira ou falsa, o comando é encerrado com a atribuição do conceito "C" (expressão verdadeira) ou "D" (expressão falsa).

A compreensão do funcionamento dos comandos de seleção aninhados é bem mais clara do que a da sequência de comandos condicionais (opção 1). Além disso, a segunda opção de representação realiza menos comparações do que a primeira, o que diminui o tempo de execução.

uso de indentação para delimitar comandos aninhados. A pseudolinguagem utilizada neste livro faz uso da indentação para mostrar visualmente o escopo de cada um dos coman-

dos de seleção aninhados. Sem a indentação, é bem mais difícil visualizar o funcionamento dos comandos aninhados, como pode ser observado na reescrita do trecho do Algoritmo 4.4 que examina a média:

```
se média ≥ 9
então conceito ← 'A'
senão se média ≥ 7,5
então conceito ← 'B'
senão se média ≥ 6,0
então conceito ← 'C'
senão conceito ← 'D'  {MÉDIA < 6}
```

Contudo, a indentação por si só não garante a correção do código e pode até mesmo mascarar erros se não corresponder à sintaxe do código utilizado. No trecho a seguir, no comando de seleção dupla, o comando da cláusula `então` é um comando condicional. A indentação utilizada faz crer que a cláusula `senão` pertence ao comando mais externo, quando, pela sintaxe, ela pertence ao mais interno:

```
se nota1 = 10      {COMANDO DE SELEÇÃO DUPLA}
então se média > 9 {COMANDO CONDICIONAL}
      então escrever ('Parabéns pela boa média!')
senão escrever ('A primeira nota não é 10! ')
```

A indentação que reflete a sintaxe do que está escrito é:

```
se nota1 = 10      {COMANDO SELEÇÃO DUPLA TRATADO COMO CONDICIONAL}
então se média > 9 {COMANDO CONDICIONAL TRATADO COMO SELEÇÃO DUPLA}
      então escrever ('Parabéns pela boa média!')
      senão escrever ('A primeira nota não é 10!')
```

Da forma como está o trecho, independentemente da indentação utilizada, se a `Nota1` fornecida for 10 e a média não for superior a 9, será produzida a mensagem 'A primeira nota não é 10!', que claramente está incorreta. Nesse caso, o problema pode ser corrigido através do uso dos delimitadores de um comando composto, para indicar que somente a cláusula `então` faz parte do comando que testa a condição "`se média > 9`":

```
se nota1 = 10      {COMANDO DE SELEÇÃO DUPLA}
então início       {COMANDO COMPOSTO}
      se média > 9 {COMANDO CONDICIONAL}
      então escrever ('Parabéns pela boa média! )
      fim
senão escrever ('A primeira nota não é 10!')
```

Parte 7

Comandos Condicionais Compostos (Pseudocódigo)

4.3 comando de seleção dupla

Supondo que, além de informar a média das três notas do aluno, também se queira que o programa informe se ele foi aprovado (quando a média for igual ou superior a 6) ou reprovado (média inferior a 6). Dois comandos, `se-então`, são necessários para imprimir essas mensagens:

```
se média ≥ 6
então escrever('Aprovado')      {INFORMA SE O ALUNO FOI APROVADO}
se média < 6
então escrever('Reprovado')     {INFORMA SE O ALUNO FOI REPROVADO}
```

Os dois comandos implementam ações mutuamente exclusivas e dependem da avaliação de uma mesma condição, sendo uma das ações associada ao resultado verdadeiro e outra ao resultado falso. Para evitar a repetição da comparação, pode ser utilizado um **comando de seleção dupla** que, a partir do resultado da avaliação de uma condição, seleciona um de dois comandos para ser executado. Sua sintaxe é:

```
se <expressão lógica>
então <comando>
senão <comando>
```

Somente um comando pode ser definido em cada uma das cláusulas `então` e `senão`. Esse comando pode ser simples ou composto, como no caso do comando de seleção simples. O exemplo anterior, resolvido através de dois comandos de seleção simples, equivale ao seguinte comando de seleção dupla:

```
se média ≥ 6
então escrever('Aprovado')
senão escrever('Reprovado')
```

O fluxograma que representa esse comando, mostrado na Figura 4.4, mostra claramente que o fluxo do programa passa por apenas um dos dois comandos, o qual é selecionado pelo resultado da expressão lógica.

O algoritmo que calcula a média de três notas e informa se o aluno foi aprovado ou reprovado é o seguinte:

```
Algoritmo 4.3 – Média4
{INFORMA A MÉDIA DO ALUNO E SE FOI APROVADO OU REPROVADO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          {Informação de aprovado ou reprovado}
início
  ler (nota1, nota2, nota3)     {ENTRADA DAS 3 NOTAS}
  média ← (nota1+nota2+nota3)/3
  escrever (média)              {INFORMA MÉDIA CALCULADA}
```

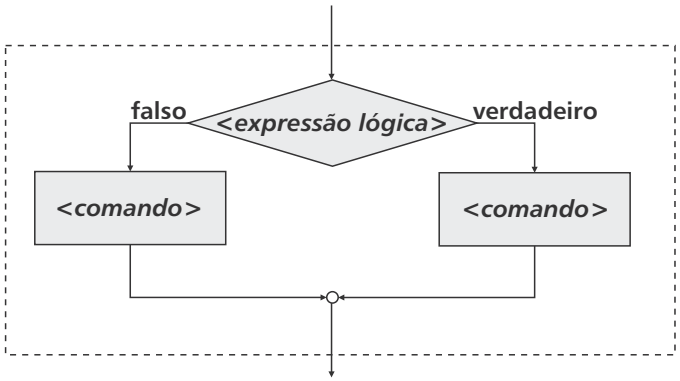


figura 4.4 Fluxograma do comando de seleção dupla.

```
  se média ≥ 6
  então escrever('Aprovado')    {INFORMA SE O ALUNO FOI APROVADO}
  senão escrever('Reprovado')   {INFORMA SE O ALUNO FOI REPROVADO}
fim
```

4.4 → comandos de seleção aninhados

Conforme visto, somente um comando pode ser utilizado nas cláusulas `então` e `senão`, mas não há restrição quanto ao tipo de comando. Pode, inclusive, ser usado um novo comando de seleção – simples ou dupla. Nesse caso, diz-se que os comandos de seleção estão aninhados ou encadeados.

Estendendo um pouco mais a aplicação de apoio a um professor, suponha que se queira obter o conceito do aluno com base na sua média, de acordo com a seguinte conversão:

Conceito A: Média ≥ 9,0
Conceito B: 9,0 > Média ≥ 7,5
Conceito C: 7,5 > Média ≥ 6,0
Conceito D: Média < 6,0

A solução pode ser implementada através de uma sequência de comandos condicionais (opção 1):

```
se média ≥ 9
então conceito ← 'A'
se (média < 9) e (média ≥ 7,5)
então conceito ← 'B'
se (média < 7,5) e (média ≥ 6,0)
então conceito ← 'C'
se (média < 6)
então conceito ← 'D'
```

Nessa sequência de comandos, somente uma das condições será verdadeira e, apesar disso, todas as condições serão sempre avaliadas, desnecessariamente. Para evitar isso, o algoritmo a seguir calcula a média e o conceito, utilizando comandos de seleção dupla aninhados (opção 2). Note que, uma vez encontrada uma condição verdadeira, as que estão após ela, na cláusula senão, não são mais avaliadas. Cabe ressaltar que, nessa solução, não foi feita a análise da validade dos dados de entrada, partindo-se do pressuposto que eles foram corretamente informados.

```
Algoritmo 4.4 - MédiaConceito1
{INFORMA A MÉDIA E O CONCEITO DE UM ALUNO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          conceito (caractere)
início
  ler (nota1, nota2, nota3)      {ENTRADA DAS 3 NOTAS}
  média ← (nota1+nota2+nota3)/3 {CÁLCULO DA MÉDIA}
  escrever (média)               {INFORMA MÉDIA CALCULADA}
  se média ≥ 9                   {CÁLCULO DO CONCEITO}
  então conceito ← 'A'
  senão se média ≥ 7,5
        então conceito ← 'B'
        senão se média ≥ 6,0
              então conceito ← 'C'
              senão conceito ← 'D'  {MÉDIA < 6}
  escrever (conceito)            {INFORMA CONCEITO}
fim
```

Nesse algoritmo, o trecho de programa que calcula o conceito corresponde a um único comando de seleção dupla. Se a média for igual ou superior a 9,0, o conceito "A" é atribuído ao aluno e a execução desse comando termina. No caso dessa condição não ser verdadeira, então é avaliada a segunda condição, que verifica se a média é igual ou superior a 7,5. Se essa condição for verdadeira, o aluno recebe o conceito "B" e o comando é concluído. Se não for verdadeira, então a média é novamente analisada, dessa vez verificando se é maior ou igual a 6,0. Finalmente, independentemente da condição ser verdadeira ou falsa, o comando é encerrado com a atribuição do conceito "C" (expressão verdadeira) ou "D" (expressão falsa).

A compreensão do funcionamento dos comandos de seleção aninhados é bem mais clara do que a da sequência de comandos condicionais (opção 1). Além disso, a segunda opção de representação realiza menos comparações do que a primeira, o que diminui o tempo de execução.

uso de indentação para delimitar comandos aninhados. A pseudolinguagem utilizada neste livro faz uso da indentação para mostrar visualmente o escopo de cada um dos coman-

dos de seleção aninhados. Sem a indentação, é bem mais difícil visualizar o funcionamento dos comandos aninhados, como pode ser observado na reescrita do trecho do Algoritmo 4.4 que examina a média:

```
se média ≥ 9
então conceito ← 'A'
senão se média ≥ 7,5
então conceito ← 'B'
senão se média ≥ 6,0
então conceito ← 'C'
senão conceito ← 'D'  {MÉDIA < 6}
```

Contudo, a indentação por si só não garante a correção do código e pode até mesmo mascarar erros se não corresponder à sintaxe do código utilizado. No trecho a seguir, no comando de seleção dupla, o comando da cláusula `então` é um comando condicional. A indentação utilizada faz crer que a cláusula `senão` pertence ao comando mais externo, quando, pela sintaxe, ela pertence ao mais interno:

```
se nota1 = 10      {COMANDO DE SELEÇÃO DUPLA}
então se média > 9 {COMANDO CONDICIONAL}
      então escrever ('Parabéns pela boa média!')
senão escrever ('A primeira nota não é 10! ')
```

A indentação que reflete a sintaxe do que está escrito é:

```
se nota1 = 10      {COMANDO SELEÇÃO DUPLA TRATADO COMO CONDICIONAL}
então se média > 9 {COMANDO CONDICIONAL TRATADO COMO SELEÇÃO DUPLA}
      então escrever ('Parabéns pela boa média!')
      senão escrever ('A primeira nota não é 10!')
```

Da forma como está o trecho, independentemente da indentação utilizada, se a `Nota1` fornecida for 10 e a média não for superior a 9, será produzida a mensagem 'A primeira nota não é 10!', que claramente está incorreta. Nesse caso, o problema pode ser corrigido através do uso dos delimitadores de um comando composto, para indicar que somente a cláusula `então` faz parte do comando que testa a condição "`se média > 9`":

```
se nota1 = 10      {COMANDO DE SELEÇÃO DUPLA}
então início       {COMANDO COMPOSTO}
      se média > 9 {COMANDO CONDICIONAL}
      então escrever ('Parabéns pela boa média! )
      fim
senão escrever ('A primeira nota não é 10!')
```

exercício 4.2 Dados os coeficientes de uma equação do 2º grau, calcular e informar os valores de suas raízes.

```
Algoritmo EquaçãoSegundoGrau
{INFORMA OS VALORES DAS RAÍZES DE UMA EQUAÇÃO DO SEGUNDO GRAU}
  Entradas: a, b, c (real)              {COEFICIENTES DA EQUAÇÃO}
  Saídas: r1, r2 (real)                 {RAÍZES}
  Variável auxiliar: disc (real)        {DISCRIMINANTE}
início
  ler(a, b, c)  {ENTRADA DOS VALORES DOS COEFICIENTES DA EQUAÇÃO}
  se a = 0
  então início
        escrever('Não é equação do segundo grau! ')
        escrever('Raiz = ', (- c / b ))
        fim
  senão início
        disc ← sqr(b) - 4 * a * c       {CÁLCULO DO DISCRIMINANTE}
        se disc < 0
        então escrever('Raízes imaginárias !')
        senão início
              r1 ← ( - b + sqrt ( disc ) ) / ( 2 * a )
              r2 ← ( - b - sqrt ( disc ) ) / ( 2 * a )
              escrever('Raízes: ', r1, r2)
              fim
       fim
fim
```

exercício 4.3 Processar uma venda de livros em uma livraria. O cliente poderá comprar diversas unidades de um mesmo tipo de livro. O código que define o tipo do livro vendido (A, B, C) e o número de unidades desse livro são fornecidos como dados de entrada.

Preços: Tipo A – R$ 10,00
Tipo B – R$ 20,00
Tipo C – R$ 30,00

Calcular e informar o preço a pagar. Caso tenham sido vendidos mais de 10 livros, exibir uma mensagem informando isso.

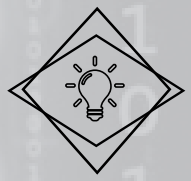
A solução deste problema é mostrada em duas etapas. Inicialmente, é apresentado um algoritmo em passos gerais para dar uma visão global da solução. Depois, cada um dos passos é detalhado, dando origem ao algoritmo completo.

```
Algoritmo UmaVenda - PASSOS GERAIS
{PROCESSA UMA VENDA DE LIVROS}
  Entradas: tipo do livro ('A', 'B' ou 'C')
            número de livros
  Saídas: preço a pagar
          mensagem indicando que foram vendidas mais de 10 unidades
```

1. Obter dados
2. Calcular preço de venda
3. Emitir mensagem caso necessário
4. Terminar

Detalhamento do algoritmo:

```
Algoritmo UmaVenda
{PROCESSA UMA VENDA DE LIVROS}
  Entradas: código (caractere)          {CÓDIGO DO LIVRO}
            numeroUnidades (inteiro)    {NR. UNIDADES VENDIDAS}
  Saídas: aPagar (real)                 {PREÇO A PAGAR}
          {Mensagem indicando que foram vendidas mais de 10
           unidades}
início
  ler(código, numeroUnidades)           {ENTRADA DE DADOS}
  se código = 'A'                       {CÁLCULO DO PREÇO DA VENDA}
  então aPagar ← numeroUnidades * 10
  senão se código = 'B'
        então aPagar ← numeroUnidades * 20
        senão se código = 'C'
              então aPagar ← numeroUnidades * 30
              senão início              {CÓDIGO ESTÁ INCORRETO}
                    aPagar ← 0
                    escrever('Código errado')
                    fim
  se aPagar > 0                         {CÓDIGO ERA VÁLIDO}
  então início
        escrever(aPagar)                {INFORMA VALOR A PAGAR}
        se numeroUnidades > 10
        então escrever ('Foram vendidas mais de 10 unidades')
        fim
  fim
```



INOVAÇÃO
E TECNOLOGIA

unidade
2

Parte 8

Comandos Condicionais de Múltipla Escolha

4.5 comando de seleção múltipla

O **comando de seleção múltipla** seleciona uma dentre diversas opções, com base na avaliação de uma expressão. Na pseudolinguagem aqui utilizada, a sintaxe deste comando é:

```
caso <expressão> seja
  <rótulo 1>: <comando 1>
  <rótulo 2>: <comando 2>
        ...
  <rótulo n>: <comando n>
  [ senão <comandos> ]
fim caso
```

onde os símbolos "[" e "]" significam que essa linha é opcional.

O comando inicia com um cabeçalho `caso <expressão> seja`, seguido de uma série de comandos rotulados, ou seja, comandos precedidos por um valor seguido do caractere ":". O resultado da avaliação da expressão utilizada no cabeçalho e os valores representados nos rótulos devem ser todos do mesmo tipo e corresponder a um valor ordinal como, por exemplo, inteiro ou caractere. Cada rótulo corresponde a somente um comando. Um comando composto pode ser utilizado caso se queira executar mais de uma ação para um determinado rótulo.

Depois de avaliada a expressão, seu resultado é comparado com cada um dos rótulos, na ordem em que são definidos. Somente o comando associado ao primeiro rótulo que for igual ao resultado da expressão é executado. Só a igualdade é verificada. Se o valor da expressão não for igual a qualquer dos rótulos, nada será executado pelo comando. Opcionalmente, poderá ser utilizada a cláusula `senão`, que indica o comando a ser executado caso nenhum dos rótulos corresponda ao valor da expressão do cabeçalho. O final do comando de seleção múltipla é indicado pelas palavras reservadas `fim caso`. Por exemplo, supondo que a variável `a` seja uma variável do tipo inteiro:

```
caso a seja
   1: a ← 0          {COMANDO SIMPLES PARA O CASO a = 1}
   2: início         {COMANDO COMPOSTO PARA O CASO a = 2}
      ler(a)
      escrever(a)
      fim
   3: a ← a + 1      {COMANDO SIMPLES PARA O CASO a = 3}
   senão escrever(a){CLÁUSULA OPCIONAL PARA OUTROS VALORES DE a}
fim caso
```

Um comando de seleção múltipla equivale a um comando de seleção dupla com outros comandos de seleção dupla aninhados nele. O exemplo anterior equivale a:

```
se a = 1
então a ← 0                    {COMANDO SIMPLES PARA O CASO a = 1}
senão se a = 2
      então início             {COMANDO COMPOSTO PARA O CASO a = 2}
            ler(a)
            escrever(a)
            fim
      senão se a = 3
            então a ← a + 1    {COMANDO SIMPLES PARA O CASO a = 3}
            senão escrever(a)  {CLÁUSULA OPCIONAL}
```

Uma vantagem de usar o comando de seleção múltipla em lugar desses comandos aninhados está na possibilidade de utilizar somente um nível de indentação, o que torna mais clara a visualização das alternativas existentes.

A Figura 4.5 representa o fluxograma correspondente ao exemplo anterior. Pode-se observar que o fluxo do programa somente passará por um dos comandos associados aos rótulos.

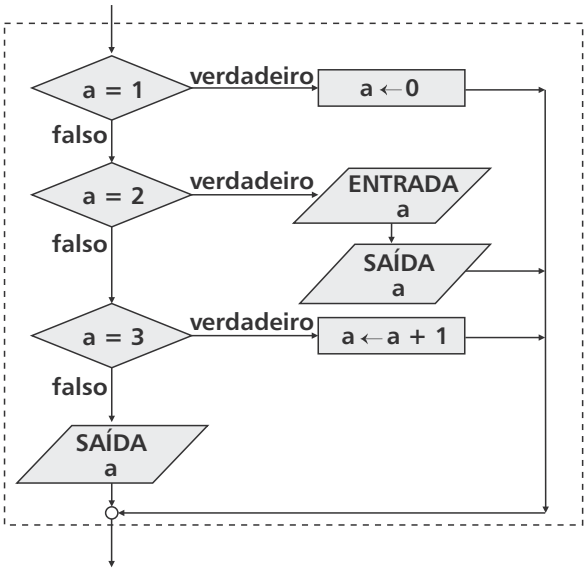


figura 4.5 Fluxograma de um comando de seleção múltipla.

O algoritmo a seguir, que calcula e informa a média e o conceito de um aluno, ilustra a utilização de rótulos do tipo caractere em um comando de seleção múltipla:

```
Algoritmo 4.5 - MédiaConceito2
{INFORMA MÉDIA E CONCEITO DE UM ALUNO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          Informação do conceito do aluno
  Variável auxiliar: conceito (char)
```

```
início
  ler (nota1, nota2, nota3)                          {ENTRADA DAS 3 NOTAS}
  média ← (nota1 + nota2 + nota3) / 3                {CALCULA MÉDIA}
  escrever (média)                                   {INFORMA MÉDIA}
  se média ≥ 9                                       {CÁLCULO DO CONCEITO}
  então conceito ← 'A'
  senão se média ≥ 7,5
        então conceito ← 'B'
        senão se média ≥ 6,0
              então conceito ← 'C'
              senão conceito ← 'D' {MÉDIA < 6}
  caso conceito seja                                 {INFORMA CONCEITO}
     'A': escrever('Conceito A – Parabéns!')
     'B': escrever('Conceito B')
     'C': escrever('Conceito C')
     'D': escrever('Conceito D – Você foi reprovado')
  fim caso
 fim
```

A implementação do comando de seleção múltipla varia, dependendo da linguagem de programação utilizada. Algumas linguagens permitem que um comando seja rotulado com uma lista de valores, ou mesmo com um intervalo. No exemplo a seguir, `nota` somente pode conter um valor inteiro:

```
caso nota seja
            0..5: escrever('Reprovado'){RÓTULO DO TIPO INTERVALO}
   6, 7, 8, 9, 10: escrever('Aprovado') {LISTA DE RÓTULOS}
fim caso
```

O primeiro comando é rotulado com o intervalo `0..5`, representando os valores inteiros `0`, `1`, `2`, `3`, `4` e `5`. O comando associado ao intervalo será executado quando o valor da variável `nota` for um desses valores. O segundo comando apresenta uma lista de rótulos. Se o valor da variável `nota` for igual a um deles, o segundo comando será executado. Observar que a utilização desses dois tipos de rótulos gerou um comando conciso e muito fácil de compreender.

Parte 9

Comandos Condicionais de Múltipla Escolha (Pseudocódigo)

4.5 comando de seleção múltipla

O **comando de seleção múltipla** seleciona uma dentre diversas opções, com base na avaliação de uma expressão. Na pseudolinguagem aqui utilizada, a sintaxe deste comando é:

```
caso <expressão> seja
  <rótulo 1>: <comando 1>
  <rótulo 2>: <comando 2>
        ...
  <rótulo n>: <comando n>
  [ senão <comandos> ]
fim caso
```

onde os símbolos "[" e "]" significam que essa linha é opcional.

O comando inicia com um cabeçalho `caso <expressão> seja`, seguido de uma série de comandos rotulados, ou seja, comandos precedidos por um valor seguido do caractere ":". O resultado da avaliação da expressão utilizada no cabeçalho e os valores representados nos rótulos devem ser todos do mesmo tipo e corresponder a um valor ordinal como, por exemplo, inteiro ou caractere. Cada rótulo corresponde a somente um comando. Um comando composto pode ser utilizado caso se queira executar mais de uma ação para um determinado rótulo.

Depois de avaliada a expressão, seu resultado é comparado com cada um dos rótulos, na ordem em que são definidos. Somente o comando associado ao primeiro rótulo que for igual ao resultado da expressão é executado. Só a igualdade é verificada. Se o valor da expressão não for igual a qualquer dos rótulos, nada será executado pelo comando. Opcionalmente, poderá ser utilizada a cláusula `senão`, que indica o comando a ser executado caso nenhum dos rótulos corresponda ao valor da expressão do cabeçalho. O final do comando de seleção múltipla é indicado pelas palavras reservadas `fim caso`. Por exemplo, supondo que a variável `a` seja uma variável do tipo inteiro:

```
caso a seja
   1: a ← 0          {COMANDO SIMPLES PARA O CASO a = 1}
   2: início         {COMANDO COMPOSTO PARA O CASO a = 2}
      ler(a)
      escrever(a)
      fim
   3: a ← a + 1      {COMANDO SIMPLES PARA O CASO a = 3}
   senão escrever(a) {CLÁUSULA OPCIONAL PARA OUTROS VALORES DE a}
fim caso
```

Um comando de seleção múltipla equivale a um comando de seleção dupla com outros comandos de seleção dupla aninhados nele. O exemplo anterior equivale a:

```
se a = 1
então a ← 0                    {COMANDO SIMPLES PARA O CASO a = 1}
senão se a = 2
      então início             {COMANDO COMPOSTO PARA O CASO a = 2}
            ler(a)
            escrever(a)
            fim
      senão se a = 3
            então a ← a + 1    {COMANDO SIMPLES PARA O CASO a = 3}
            senão escrever(a)  {CLÁUSULA OPCIONAL}
```

Uma vantagem de usar o comando de seleção múltipla em lugar desses comandos aninhados está na possibilidade de utilizar somente um nível de indentação, o que torna mais clara a visualização das alternativas existentes.

A Figura 4.5 representa o fluxograma correspondente ao exemplo anterior. Pode-se observar que o fluxo do programa somente passará por um dos comandos associados aos rótulos.

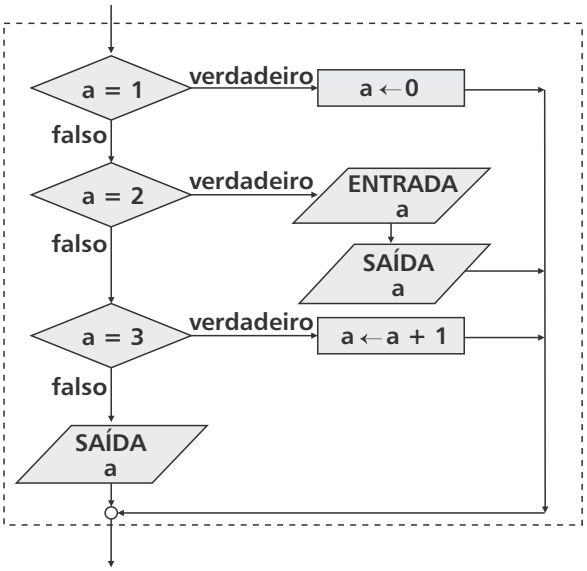


figura 4.5 Fluxograma de um comando de seleção múltipla.

O algoritmo a seguir, que calcula e informa a média e o conceito de um aluno, ilustra a utilização de rótulos do tipo caractere em um comando de seleção múltipla:

```
Algoritmo 4.5 – MédiaConceito2
{INFORMA MÉDIA E CONCEITO DE UM ALUNO}
  Entradas: nota1, nota2, nota3 (real)
  Saídas: média (real)
          Informação do conceito do aluno
  Variável auxiliar: conceito (char)
```

```
início
  ler (nota1, nota2, nota3)                          {ENTRADA DAS 3 NOTAS}
  média ← (nota1 + nota2 + nota3) / 3                {CALCULA MÉDIA}
  escrever (média)                                   {INFORMA MÉDIA}
  se média ≥ 9                                       {CÁLCULO DO CONCEITO}
  então conceito ← 'A'
  senão se média ≥ 7,5
        então conceito ← 'B'
        senão se média ≥ 6,0
              então conceito ← 'C'
              senão conceito ← 'D' {MÉDIA < 6}
  caso conceito seja                                 {INFORMA CONCEITO}
     'A': escrever('Conceito A - Parabéns!')
     'B': escrever('Conceito B')
     'C': escrever('Conceito C')
     'D': escrever('Conceito D - Você foi reprovado')
  fim caso
 fim
```

A implementação do comando de seleção múltipla varia, dependendo da linguagem de programação utilizada. Algumas linguagens permitem que um comando seja rotulado com uma lista de valores, ou mesmo com um intervalo. No exemplo a seguir, `nota` somente pode conter um valor inteiro:

```
caso nota seja
            0..5: escrever('Reprovado'){RÓTULO DO TIPO INTERVALO}
   6, 7, 8, 9, 10: escrever('Aprovado') {LISTA DE RÓTULOS}
fim caso
```

O primeiro comando é rotulado com o intervalo `0..5`, representando os valores inteiros `0`, `1`, `2`, `3`, `4` e `5`. O comando associado ao intervalo será executado quando o valor da variável `nota` for um desses valores. O segundo comando apresenta uma lista de rótulos. Se o valor da variável `nota` for igual a um deles, o segundo comando será executado. Observar que a utilização desses dois tipos de rótulos gerou um comando conciso e muito fácil de compreender.

4.6 exercícios de fixação

exercício 4.1 Ler um número inteiro. Se o número lido for positivo, escrever uma mensagem indicando se ele é par ou ímpar.

```
Algoritmo ParOuÍmpar
{INFORMA SE UM VALOR LIDO DO TECLADO É PAR OU ÍMPAR}
  Entrada: valor (inteiro)            {VALOR A SER TESTADO}
  Saída: Mensagem de 'par' ou 'ímpar'
```

```
  Variável auxiliar: ehPar (lógica)
início
  ler(valor)                        {ENTRADA DO VALOR A SER TESTADO}
  se  valor ≥ 0                     {VALOR LIDO É POSITIVO}
  então início
        ehPar ← (valor mod 2) = 0 {VERIFICA SE VALOR É PAR}
        se ehPar                    {VERDADEIRO SE ehPar FOR VERDADEIRO}
        então escrever('Par ! ')
        senão escrever('Ímpar ! ')
        fim
fim
```

exercício 4.2 Dados os coeficientes de uma equação do 2º grau, calcular e informar os valores de suas raízes.

```
Algoritmo EquaçãoSegundoGrau
{INFORMA OS VALORES DAS RAÍZES DE UMA EQUAÇÃO DO SEGUNDO GRAU}
  Entradas: a, b, c (real)              {COEFICIENTES DA EQUAÇÃO}
  Saídas: r1, r2 (real)                 {RAÍZES}
  Variável auxiliar: disc (real)        {DISCRIMINANTE}
início
  ler(a, b, c)  {ENTRADA DOS VALORES DOS COEFICIENTES DA EQUAÇÃO}
  se a = 0
  então início
        escrever('Não é equação do segundo grau! ')
        escrever('Raiz = ', (- c / b ))
        fim
  senão início
        disc ← sqr(b) - 4 * a * c       {CÁLCULO DO DISCRIMINANTE}
        se disc < 0
        então escrever('Raízes imaginárias !')
        senão início
              r1 ← ( - b + sqrt ( disc ) ) / ( 2 * a )
              r2 ← ( - b - sqrt ( disc ) ) / ( 2 * a )
              escrever('Raízes: ', r1, r2)
              fim
       fim
fim
```

exercício 4.3 Processar uma venda de livros em uma livraria. O cliente poderá comprar diversas unidades de um mesmo tipo de livro. O código que define o tipo do livro vendido (A, B, C) e o número de unidades desse livro são fornecidos como dados de entrada.

Preços: Tipo A – R$ 10,00
Tipo B – R$ 20,00
Tipo C – R$ 30,00

Calcular e informar o preço a pagar. Caso tenham sido vendidos mais de 10 livros, exibir uma mensagem informando isso.

A solução deste problema é mostrada em duas etapas. Inicialmente, é apresentado um algoritmo em passos gerais para dar uma visão global da solução. Depois, cada um dos passos é detalhado, dando origem ao algoritmo completo.

```
Algoritmo UmaVenda - PASSOS GERAIS
{PROCESSA UMA VENDA DE LIVROS}
  Entradas: tipo do livro ('A', 'B' ou 'C')
            número de livros
  Saídas: preço a pagar
          mensagem indicando que foram vendidas mais de 10 unidades
```

1. Obter dados
2. Calcular preço de venda
3. Emitir mensagem caso necessário
4. Terminar

Detalhamento do algoritmo:

```
Algoritmo UmaVenda
{PROCESSA UMA VENDA DE LIVROS}
  Entradas: código (caractere)         {CÓDIGO DO LIVRO}
            numeroUnidades (inteiro)   {NR. UNIDADES VENDIDAS}
  Saídas: aPagar (real)                {PREÇO A PAGAR}
          {Mensagem indicando que foram vendidas mais de 10
           unidades}
início
  ler(código, numeroUnidades)          {ENTRADA DE DADOS}
  se código = 'A'                      {CÁLCULO DO PREÇO DA VENDA}
  então aPagar ← numeroUnidades * 10
  senão se código = 'B'
        então aPagar ← numeroUnidades * 20
        senão se código = 'C'
              então aPagar ← numeroUnidades * 30
              senão início             {CÓDIGO ESTÁ INCORRETO}
                    aPagar ← 0
                    escrever('Código errado')
                    fim
  se aPagar > 0                        {CÓDIGO ERA VÁLIDO}
  então início
        escrever(aPagar)               {INFORMA VALOR A PAGAR}
        se numeroUnidades > 10
        então escrever ('Foram vendidas mais de 10 unidades')
        fim
  fim
```

exercício 4.4 Processar uma venda em um estabelecimento comercial. São fornecidos o código do produto vendido e seu preço. Deverá ser dado um desconto, de acordo com a seguinte tabela:

Código A – 20%
Código B – 10%

Produtos com código diferente de A ou B não terão desconto. Além disso, se o total a pagar for maior ou igual a R$ 80,00, deverá ser dado um desconto adicional de 10%. Calcular e informar o preço a pagar e o desconto dado na compra, se for o caso.

```
Algoritmo UmaVendaComércio
{PROCESSA UMA VENDA EM UM ESTABELECIMENTO COMERCIAL}
   Entradas: preço (real)
             código (caractere)
   Saídas: desconto (real)
           aPagar (real)
início
  ler(código, preço)              {ENTRADA DE DADOS}
  aPagar ← preço                  {INICIALIZA aPagar}
  desconto ← 0                    {INICIALIZA desconto}
  se código = 'A'                 {CALCULA desconto}
  então desconto ← preço / 5
  senão se código = 'B'
        então desconto ← preço / 10
  aPagar ← aPagar - desconto      {CALCULA aPagar}
  se aPagar ≥ 80,00               {VERIFICA SE COMPRA ≥ 80,00}
  então início
        desconto ← desconto + aPagar / 10
        aPagar ← aPagar * 0,9     {MAIS 10% DE DESCONTO}
        fim
  escrever(aPagar)                {INFORMA VALOR A PAGAR}
  se desconto ≠ 0
  então escrever(desconto)        {INFORMA DESCONTO RECEBIDO}
  fim
```

exercício 4.5 Escrever um programa que, dados um determinado mês (representado por um número inteiro) e um ano, informe quantos dias tem esse mês. Para determinar o número de dias de fevereiro, verificar primeiro se o ano é bissexto. Um ano será bissexto se terminar em 00 e for divisível por 400, ou se não terminar por 00, mas for divisível por 4.

```
Algoritmo DiasDoMês1
{INFORMA QUANTOS DIAS TEM UM DETERMINADO MÊS}
  Entrada: mês, ano (inteiro)   {DADOS DE ENTRADA - MÊS E ANO}
  Saída: dias (inteiro)         {NÚMERO DE DIAS DO MÊS NESTE ANO}
  Variável auxiliar: ehBissexto (lógica)
```

```
início
  ler(mês, ano)        {ENTRADA DE DADOS}
  se mês = 2           {SE FEVEREIRO, VERIFICA SE ANO EH BISSEXTO}
  início
     eh Bissexto  ← falso
     se (ano mod 100) = 0
     então início
           se (ano mod 400) = 0
           então ehBissexto  ←  verdadeiro
           fim
     senão se (ano mod 4) = 0
           então ehBissexto  ←  verdadeiro
  fim
  se mês = 2                    {FEVEREIRO}
  então se ehBissexto
        então dias ← 29
        senão dias ← 28
  senão se (mês=4) ou (mês=6) ou (mês=9) ou (mês=11)
        então dias ← 30         {ABRIL, JUNHO, SETEMBRO, NOVEMBRO}
        senão dias ← 31         {DEMAIS MESES}
  escrever(dias)                {INFORMA NÚMERO DE DIAS DO MÊS}
fim
```

Algoritmo DiasDoMês2

```
{INFORMA QUANTOS DIAS TEM UM DETERMINADO MÊS}
  Entrada: mês, ano (inteiro)   {DADOS DE ENTRADA - MÊS E ANO}
  Saída: dias (inteiro)         {NÚMERO DE DIAS DO MÊS NESTE ANO}
  Variável auxiliar: ehBissexto (lógica)
início
  ler(mês, ano)                 {ENTRADA DE DADOS}
  caso mês seja
    2 : {FEVEREIRO}
        eh_Bissexto ← falso
        se (ano mod 100) = 0
        então início
              se (ano mod 400) = 0
              então ehBissexto  ←  verdadeiro
              fim
        senão se (ano mod 4) = 0
              então ehBissexto  ←  verdadeiro
        se ehBissexto
        então dias ← 29         {ANO BISSEXTO}
        senão dias ← 28
    4, 6, 9, 11 : dias ← 30     {ABRIL, JUNHO, SETEMBRO, NOVEMBRO}
```

```
    senão dias ← 31              {DEMAIS MESES}
 fim caso                        {FIM COMANDO DE SELEÇÃO MÚLTIPLA}
  escrever(dias)                 {INFORMA NÚMERO DE DIAS DO MÊS}
fim
```

exercício 4.6 Fazer um programa que leia um caractere e exiba uma mensagem informando se ele é uma letra, um dígito numérico ou um caractere de operação aritmética. Se o caractere não for de qualquer desses três tipos, informar que se trata de um caractere desconhecido.

```
Algoritmo IdentificarCaractere
{INFORMA TIPO DE CARACTERE LIDO}
  Entradas: lido (caractere)      {CARACTERE A SER IDENTIFICADO}
  Saídas: Mensagem indicando o tipo de caractere lido
início
  ler(lido)                       {CARACTERE A SER IDENTIFICADO}
  caso lido seja
    'A'..'Z', 'a'..'z' : escrever('Letra')
    '0'..'9': escrever('Dígito')
    '+', '-', '*', '/' : escrever('Operador aritmético')
    senão escrever('Caractere desconhecido')
  fim caso
fim
```

Sugestão: resolva este exercício sem utilizar o comando de seleção múltipla.

Parte 10

Ordenação de Dados

INTRODUÇÃO

Olá!

Neste texto você vai estudar os principais algoritmos de ordenação de dados existentes na literatura. Primeiramente, serão abordados esses algoritmos de forma conceitual, a fim de prover o entendimento da importância e a utilidade dos algoritmos de ordenação de dados. Posteriormente será apresentada uma análise do funcionamento peculiar de cada um dos algoritmos, ilustrando o seu funcionamento através de exemplos didáticos. Por fim, ainda neste capítulo, serão apresentadas estratégias para identificação do algoritmo de ordenação de dados mais adequado a cada contexto.

OBJETIVOS DE APRENDIZAGEM

Ao final desta unidade você deve apresentar os seguintes aprendizados:

- Diferenciar alguns dos métodos de ordenação interna de dados.
- Reconhecer o funcionamento de alguns dos métodos de ordenação interna de dados.
- Identificar os benefícios da aplicação da ordenação de dados na solução de problemas do dia a dia.

PRINCIPAIS ALGORITMOS DE ORDENAÇÃO DE DADOS

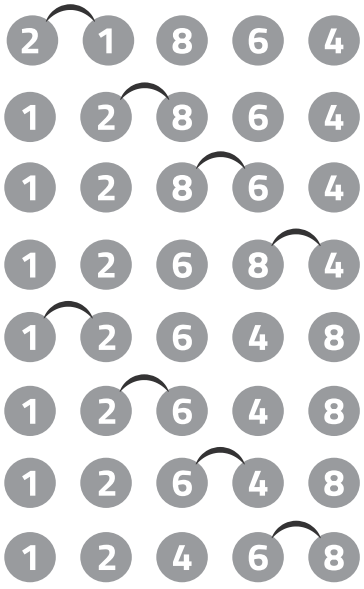
Ao visualizarmos a pluralidade na sentença algoritmos de ordenação de dados, podemos facilmente nos questionar sobre o motivo da existência de mais de um algoritmo destinado ao mesmo fim. Isto porque cada algoritmo possui suas características peculiares, contudo, todos têm um único e principal objetivo, que é ordenar de maneira correta os dados. Nesta seção será analisado cada um dos métodos de ordenação para que, ao final deste estudo, você esteja apto a diferenciar os métodos de ordenação de dados.

ORDENAÇÃO POR BOLHA (*BUBBLE SORT*)

Dentre todos, este é o algoritmo com o funcionamento mais simples de entender e de implementar. A sua estrutura básica consiste na comparação de um elemento com o elemento do índice imediatamente posterior e na verificação dos valores armazenados nos índices. Como exemplo, o objetivo

deste algoritmo pode ser ordenar elementos de forma crescente, da esquerda para a direita. Na comparação, caso o valor da esquerda seja maior, o algoritmo de bolha efetua a troca dos elementos: (vetor[x] < vetor[x+1]).

Este algoritmo é indicado apenas para situações em que o arquivo que se deseja ordenar for pequeno, pois, devido a este algoritmo não otimizar as comparações e comparar em todos os casos todos os elementos do vetor, ele tem um desempenho inferior aos demais algoritmos. Veja, na imagem a seguir, que são necessárias doze iterações para ordenar um algoritmo de 5 elementos, sendo que, caso o menor número estivesse no último elemento, seriam necessárias pelo menos 25 iterações, uma vez que a complexidade do algoritmo é exponencial.



Nova verificação ao final, para conferir se todos estão realmente ordenados...

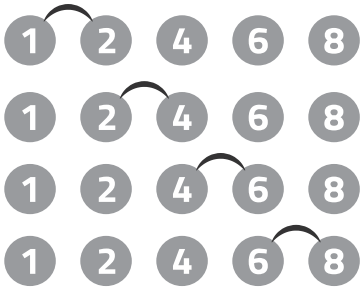


Figura 1 - Execução do algoritmo de bolha

ORDENAÇÃO POR SELEÇÃO (S*ELECTION SORT*)

Neste exemplo, o algoritmo tem por objetivo colocar o menor número na primeira posição do vetor. Diferente do *bubble sort*, que percorre todo o vetor comparando pares de elementos, o *selection sort* seleciona um elemento e o compara com todos os valores do vetor, verificando se ele é o menor. Contudo, a ordenação por seleção, assim como a ordenação por bolha, não possui um bom desempenho para grandes trabalhos de ordenação. Veja, na ilustração a seguir, as etapas necessárias para ordenar o vetor utilizando o algoritmo de ordenação por seleção.

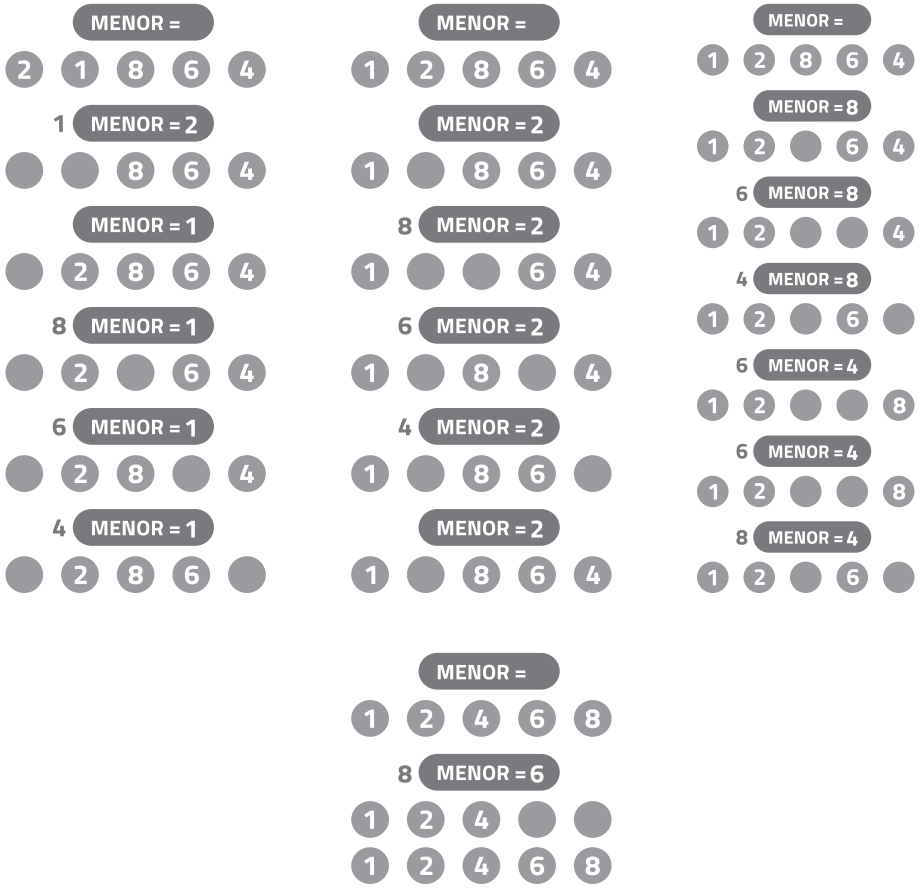
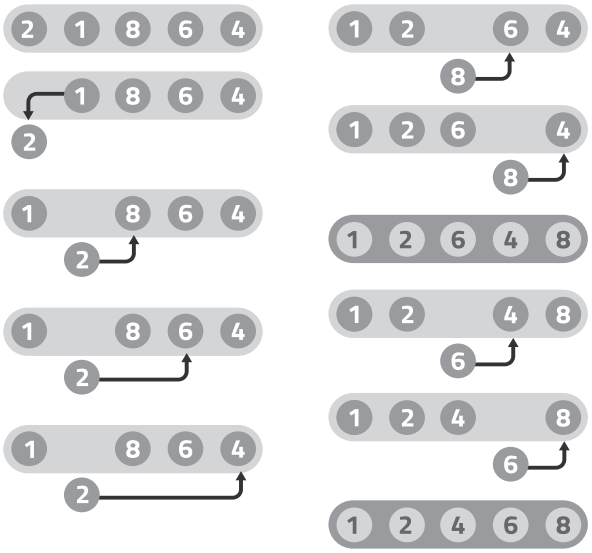


Figura 2 - Execução do algoritmo por seleção

ORDENAÇÃO RÁPIDA (*QUICKSORT*)

O algoritmo de ordenação *Quicksort*, como já diz o próprio nome, é o que possui a melhor eficiência e, por isso, maior velocidade na ordenação de grandes arquivos.



OUTROS ALGORITMOS DE ORDENAÇÃO DE DADOS

Além dos algoritmos já vistos (ordenação por bolha, ordenação por seleção e ordenação rápida), existem outros algoritmos de ordenação de dados, como o de ordenação por inserção (*Insertion Sort*) e o de ordenação por intercalação (*Merge Sort*).

***Insertion Sort*:** Consiste em inserir o elemento já em sua posição correta; dessa forma, sempre que o algoritmo verificar que o elemento do índice imediatamente posterior (x+1) é inferior ao elemento do índice imediatamente anterior (x), além de efetuar a troca entre estes dois elementos, também irá verificar se algum dos elementos anteriores é maior e, então, inserir o valor no local adequado. Por exemplo:

[5] [8] [7] [6] [3] - Vetor inicial.

[5] [8] [7] [6] [3] – O algoritmo seleciona o primeiro elemento, no caso, o número 5 e compara com o elemento imediatamente posterior, no caso, o elemento 8. Como 5 < 8, não é efetuada nenhuma troca.

[5] **[8]** [7] [6] [3] – O algoritmo seleciona o segundo elemento, no caso, o número 8 e compara com o elemento imediatamente posterior, no caso, o elemento 7. Como 8 > 7, é efetuada a troca entre eles. Contudo, antes de inserir o 7 em sua nova posição, o programa vai verificar se o elemento à esquerda do 8 é inferior ao 7; como é inferior, apenas insere o 7.

[5] [7] **[8]** [6] [3] – O algoritmo seleciona o terceiro, no caso, o número 8 e compara com o elemento imediatamente posterior, no caso, o elemento 6. Como 8 > 6, é efetuada a troca entre eles. Contudo, analisando o elemento imediatamente anterior ao 8, que é o 7, verifica-se que 7 é maior que 6 e, por isso, é feita nova troca.

[5] [6] [7] [8] [3] – O algoritmo testa o quarto elemento, no caso, o 8. Como o elemento imediatamente posterior contém o valor 3, é efetuada a troca. Contudo, o valor 3 também é inferior aos elementos anteriores ao 8 (7, 6, 3 e 5) e, por isso, são efetuadas novas trocas até que o valor 3 chegue no seu lugar. Após três novas trocas, o vetor ficará assim: [3] [5] [6] [7] [8].

Merge Sort: Este algoritmo usa a estratégia de dividir para conquistar, ou seja, divide o vetor em partes e ordena dentro de cada parte. Veja o exemplo a seguir:

[5] [3] [2] [4] [7] [1] [0] [6] - Formação inicial do vetor.

[5][3][2][4] [7][1][0][6] - O vetor é dividido em dois grupos.

[5][3] [2][4] [7] [1] [0] [6] - O vetor é novamente dividido, agora em quatro grupos, atingindo, assim, a máxima divisão possível.

[3][5] [2][4] [1][7] [0][6] – O algoritmo realiza a ordenação de cada grupo.
[3][5][2][4] [1][7][0][6] - O algoritmo realiza um reagrupamento.

[2][3][4][5] [0][1][6][7] - O algoritmo realiza a ordenação nos subgrupos maiores através da comparação dos primeiros elementos de cada grupo menor.

Veja a sequência de passos para ordenação entre os grupos:

[2] [3][4][5] **[0]** [1][6][7] - Comparação do primeiro elemento do primeiro grupo com cada elemento do segundo grupo. Se algum dos elementos do segundo grupo for menor que o primeiro elemento do primeiro grupo, é feita a troca de posição. Ao final de todas as trocas tem-se o vetor ordenado.

[0] [1] [2] [3] [4] [5] [6] [7]

ORDENAÇÃO DE DADOS – EXERCÍCIOS

1) Com base na funcionalidade dos métodos internos simples de ordenação de dados, identifique qual o método corresponde a cada alternativa descrita abaixo.

I. A _____ seleciona o menor entre os n elementos de um vetor ou uma tabela e realiza a troca deste pelo primeiro elemento. E, para o restante dos elementos, é encontrado novamente o elemento de menor chave, trocando-o pelo segundo elemento e assim por diante até chegar aos dois últimos elementos.

II. A _____ percorre elemento por elemento do vetor ou tabela, deslocando os elementos já ordenados e inserindo o elemento que deseja colocar em ordem na posição correta com relação aos elementos já ordenados.

III. A _____ é a ordenação por trocas, que envolve repetidas comparações e, se necessário, a troca de dois elementos que encontram-se um ao lado do outro. Nesse método, o elemento mais leve sobe e o mais pesado desce, ou ocorre o contrário, dependendo da ordem na qual deseja-se colocar os elementos ordenados.

Os métodos internos que representam de forma correta a sua funcionalidade descrita acima nas alternativas I, II e III são, respectivamente:

a) Ordenação por seleção / Ordenação por inserção / Ordenação bolha.
b) Ordenação por seleção / Ordenação bolha / Ordenação por inserção.
c) Ordenação por inserção / Ordenação bolha / Ordenação por seleção.
d) Ordenação bolha / Ordenação por inserção / Ordenação por seleção.
e) Ordenação por inserção / Ordenação por seleção / Ordenação bolha.

2) Em vários momentos do nosso dia a dia, precisamos de dados ordenados para agilizar nosso trabalho de pesquisa ou busca. Como exemplo, pode-se citar um relatório dos dados pessoais dos funcionários de uma empresa. Como seria consultar os dados de um funcionário, como e-mail ou telefone, se o relatório não estivesse em ordem alfabética de nome? Em função dessa necessidade de dados ordenados, existem vários métodos de ordenação, alguns melhores que outros.

Analise e julgue as alternativas a seguir, acerca dos algoritmos para ordenação interna apresentados na Unidade de Aprendizagem.

I. O algoritmo de ordenação por inserção simples apresenta um ótimo desempenho quando os elementos a serem ordenados encontram-se já inseridos de forma ordenada, não importando a quantidade de elementos a serem ordenados. Apresenta um desempenho não eficiente se os elementos encontram-se em ordem descendente/invertida.

II. Um algoritmo de ordenação é considerado estável se ele não alterar a posição relativa de elementos de mesmo valor.

III. O método bolha é um dos métodos mais fáceis de programar, mas não é eficiente comparado a outros métodos.

IV. Os métodos de ordenação simples por inserção, método bolha e por seleção possuem complexidade de O(n²) comparações.

Assinale a alternativa que contém a correta sequência de V (verdadeiro) e F (falso), correspondente às afirmativas acima.

a) V – V – F – V
b) V – F – V – F
c) F – V – F – V
d) V – V –V – F
e) V – V – V – V

3) Com relação à classificação dos métodos de ordenação e de fatores que devem ser levados em consideração no momento de avaliar e comparar os diversos métodos estudados na Unidade de Aprendizagem, avalie as alternativas apresentadas abaixo e assinale a alternativa INCORRETA.

a) Os métodos de ordenação são classificados em interno, externo e misto.

b) O número de comparações entre as chaves e o número de trocas entre os elementos para a sua ordenação não podem ser considerados uma forma de avaliar e comparar os métodos de ordenação interna estudados na Unidade porque cada algoritmo é diferente.

c) O método de ordenação é considerado estável quando, no momento da ordenação, não movimenta os elementos que são iguais, ou seja, que possuem a mesma chave.

d) O método possui um comportamento natural quando trabalha o mínimo, quando os elementos forem inseridos de forma ordenada; trabalha mais quando os elementos forem introduzidos de forma mais desordenada e trabalha o máximo quando em ordem inversa.

e) A ordenação tem por objetivo facilitar e agilizar a busca de elementos.

4) Considere o módulo de ordenação denominado ORDENA_VETOR desenvolvido em pseudocódigo:

```
procedimento ordena_vetor
var
f, indice, tmp, menor: inteiro

inicio
 para indice de 1 ate 6 passo 1 faca
    menor <- indice
    f <- indice+1
   enquanto (f <= 6) faca
        se(Elementos[j] < Elementos[menor]) entao
           menor <- f
        fimse
```

```
    f <- f+1
    fimenquanto
    tmp <- Elementos[menor];
    Elementos[menor] <- Elementos[indice];
    Elementos[indice] <- tmp;
  fimpara
fimprocedimento
```

Realize o teste de mesa para o módulo ORDENA_VETOR com os valores de entrada para o vetor "Elementos" de 6 posições, em que o vetor é uma variável local do algoritmo.

Elementos = {6,5,3,23,12,34}

Representações:

indice, f: representam os índices que controlam a posição do vetor Elementos.

Elementos: representa o vetor no qual os elementos estão armazenados.
tmp: representa uma variável auxiliar para a troca dos elementos de posição.

menor: representa a variável que armazena a posição do menor elemento encontrado.

Selecione a alternativa que representa o método de ordenação utilizado para ordenar o vetor Elementos.

a) Ordenação bolha
b) Ordenação por inserção
c) Ordenação por seleção
d) Quicksort
e) Ordenação Shell

5) O método de ordenação Quicksort é um dos métodos de ordenação interna mais eficiente em um grande número de situações práticas de aplicação em que a ordenação se fez necessária. O método apresenta diversas derivações de algoritmos para a implementação, algumas mais simples, outras utilizando a recursividade, que empregam a chamada funções de forma recursiva, ou

seja, funções que chamam por elas mesmas. Como exemplo de aplicação, podemos citar a ordenação de nomes e notas de todos os alunos de uma turma da disciplina de Cálculo dos cursos de Engenharia em ordem alfabética de forma crescente ou decrescente ou em ordem de notas, conforme necessidade do professor.

Com relação a esse método, analise as afirmações abaixo e marque qual apresenta informações INCORRETAS.

I . O método Quicksort é considerado um método de ordenação estável.

II. A escolha correta do pivô é essencial para a garantia de eficiência do algoritmo. O cálculo da mediana de três chaves é uma das formas eficientes de encontrar o elemento pivô.

III. O pivô pode ser o elemento do meio do vetor/tabela a ser ordenado.

IV. Utiliza um pivô para dividir o vetor/tabela em duas partes, em que vai colocando os menores elementos de um lado e os maiores de outro.

a) I
b) I – II
c) III – IV
d) I – II – III
e) I – II – III – IV

SAIBA+

GROZEN-SMITH, Mark. **Quicksort**. Disponível em: https://www.youtube.com/watch?v=aQiWF4E8flQ&t=12s. Acesso em: 02 jan. 2017.

SANTOS, Tiago. **Bubble Sort**. Disponível em: <https://www.youtube.com/watch?v=llX2SpDkQDc>. Acesso em: 02 jan. 2017.

SANTOS, Tiago. **Insertion Sort**. Disponível em: <https://www.youtube.com/watch?v=-Z00it6Nkz8>. Acesso em: 02 jan. 2017.

SANTOS, Thiago. **Merge Sort**. Disponível em: <https://www.youtube.com/watch?v=cDNqk4tdvqQ>. Acesso em: 02 jan. 2017.

SANTOS, Tiago. **Selection Sort**. Disponível em: <https://www.youtube.com/watch?v=BSXIolKg5F8>. Acesso em: 02 jan. 2017.

REFERÊNCIAS

HONORATO, Bruno de Almeida. **Algoritmos de Ordenação – Análise e Comparação.** Disponível em: <http://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>. Acesso em: 02 jan. 2017.

ANOTAÇÕES

CONTRIBUA COM A QUALIDADE DO SEU CURSO

Se você encontrar algum problema nesse material, entre em contato pelo email **eadproducao@unilasalle.edu.br**. Descreva o que você encontrou e indique a página.

Lembre-se: a boa educação se faz com a contribuição de todos!

UNIVERSIDADE LaSalle EAD

Av. Victor Barreto, 2288
Canoas - RS
CEP: 92010-000 | 0800 541 8500
ead@unilasalle.edu.br