algoritmos e programação

com exemplos em Pascal e C

nina edelweiss
maria aparecida castro livi

bookman

as autoras

Nina Edelweiss é engenheira eletricista e doutora em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Durante muitos anos, lecionou em cursos de Engenharia e de Ciência da Computação na UFRGS, na UFSC e na PUCRS. Foi, ainda, orientadora do Programa de Pós-Graduação em Ciência da Computação da UFRGS. É coautora de três livros, tendo publicado diversos artigos em periódicos e em anais de congressos nacionais e internacionais. Participou de diversos projetos de pesquisa financiados por agências de fomento como CNPq e FAPERGS, desenvolvendo pesquisas nas áreas de bancos de dados e desenvolvimento de software.

Maria Aparecida Castro Livi é licenciada e bacharel em Letras, e mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Desenvolveu sua carreira profissional na UFRGS, onde foi programadora e analista de sistema, antes de ingressar na carreira docente. Ministrou por vários anos a disciplina de Algoritmos e Programação para alunos dos cursos de Engenharia da Computação e Ciência da Computação. Sua área de interesse prioritário é o ensino de Linguagens de Programação, tanto de forma presencial quanto a distância.

E22a Edelweiss, Nina.
Algoritmos e programação com exemplos em Pascal e C [recurso eletrônico] / Nina Edelweiss, Maria Aparecida Castro Livi. – Dados eletrônicos. – Porto Alegre : Bookman, 2014.

Editado também como livro impresso em 2014.
ISBN 978-85-8260-190-7

1. Informática. 2. Algoritmos – Programação. I. Livi, Maria Aparecida Castro. II. Título.

CDU 004.421

Catalogação na publicação: Ana Paula M. Magnus – CRB 10/2052

capítulo 6

variáveis estruturadas: arranjos unidimensionais

Este capítulo introduz um tipo de variável estruturada denominado arranjo, que agrupa dados do mesmo tipo. Analisa os arranjos de uma dimensão, ou seja, unidimensionais, também denominados vetores. Também discute como vetores devem ser declarados e manipulados, e apresenta alguns exemplos de algoritmos de classificação e de pesquisa de dados armazenados em vetores.

Com as estruturas básicas de controle de fluxo já vistas, sequência, seleção e iteração, é possível resolver praticamente qualquer problema de forma algorítmica. Entretanto, os recursos de armazenamento até agora utilizados não são adequados para soluções que envolvam grandes volumes de dados.

Por exemplo, se um professor tem 30 alunos em uma turma e deseja calcular a média aritmética dessa turma, quantas variáveis ele necessita para ler as médias dos 30 alunos? Pode usar 30, mas uma só é suficiente. Isso porque, nesse problema, as médias dos alunos, após serem lidas e acumuladas em uma variável, não precisam mais ser guardadas. Uma única variável pode ser reaproveitada para ler todas as 30 médias. A cada nova média, ao ser utilizada a mesma variável para nova leitura, a média anterior é perdida, ou seja, "jogada no lixo", sem que isso afete o resultado pretendido, pois o valor anterior já foi acumulado para calcular a média.

Mas, se o nosso professor desejar, além de calcular e apresentar a média da turma, também listar as médias dos alunos que forem iguais ou superiores à média da turma, quantas variáveis ele precisará para armazenar os 30 valores? Nesse caso não há escolha, ele precisará de 30 variáveis. Se o professor quiser fazer o mesmo para sua turma em EAD (ensino a distância), com 150 alunos, precisará de 150 variáveis. Detalhe importante: cada variável deverá ter um nome diferente! Surge aí um grande problema: quanto maior o número de variáveis que um problema exige, maior é o número de nomes diferentes necessários. É claro que sempre dá para usar nomes de variáveis como val01, val02, val03, etc. Mas, além de trabalhosas, as soluções que seguem por esse caminho têm grande chance de serem indutoras de erros/enganos devido ao número elevado de nomes semelhantes. Se pensarmos que há problemas em que o número de valores diferentes que devem permanecer armazenados por períodos relativamente longos são na ordem de centenas e até mesmo de milhares, o uso de variáveis simples para armazenar grandes volumes de dados claramente não parece ser a melhor opção de armazenamento.

Este capítulo discute um tipo de variável estruturada denominado arranjo, que agrupa dados do mesmo tipo e possibilita trabalhar com grande volume de dados de forma eficiente. Arranjos de uma dimensão (ou seja, unidimensionais) são apresentados e analisados aqui. São vistos também alguns exemplos de algoritmos de classificação e de pesquisa para arranjos unidimensionais. Arranjos de mais dimensões serão vistos no Capítulo 7.

6.1 → arranjos

Um **arranjo** é uma variável estruturada, formada pelo agrupamento de variáveis de mesmo tipo (inteiro, real, etc.). As variáveis que integram um arranjo compartilham um único nome e são identificadas individualmente por um **índice**, que é o valor que indica sua posição no agrupamento. Os índices dos arranjos geralmente são de tipos ordinais, como inteiro e caractere, e podem ser tanto variáveis como constantes. Não existe uma vinculação obrigatória entre um arranjo e as variáveis ou constantes usadas para indexá-lo. Dessa forma, considerando um certo código, um arranjo pode ser indexado em momentos diferentes por diferentes

variáveis ou constantes, que podem igualmente ser usadas em outros momentos para a indexação de outros arranjos.

A qualquer momento pode-se acessar um determinado elemento de um arranjo, sem necessidade de acessar antes os elementos que o precedem. O índice utilizado define o elemento sendo acessado. Um elemento de um arranjo é equivalente a uma variável simples. Assim, sobre qualquer elemento de um arranjo podem ser efetuadas todas as operações até aqui vistas para variáveis simples – preencher o elemento por leitura ou por atribuição, utilizar seu valor em expressões aritméticas, assim como consultar seu valor para operações, testes ou saídas.

A ideia base dos arranjos é por vezes utilizada em situações do mundo real como, por exemplo, no caso do professor que não sabe o nome de seus alunos no início do ano letivo e então entrega a eles, ordenadamente, cartões numerados de 1 a 30, do aluno sentado na primeira fileira até o aluno sentado na última fileira. Concluída a entrega dos cartões, passa a referir-se aos alunos usando o nome genérico aluno, seguido do número que consta no cartão que o aluno recebeu, que corresponde à sua posição no conjunto dos alunos da sala: aluno 10, aluno 29, etc. O conjunto de alunos da sala constitui "o arranjo" `aluno`, em que cada aluno é indexado por sua posição (índice) na sala.

Arranjos podem ser unidimensionais (de uma só dimensão, quando precisam de um só índice para identificar seus elementos) ou multidimensionais (de duas ou mais dimensões). Os arranjos de uma só dimensão também são chamados de `vetores`, e os de duas ou mais dimensões, de `matrizes`. Este capítulo discutirá os arranjos unidimensionais, e o capítulo seguinte, os arranjos multidimensionais.

6.2 → vetores

Um **vetor** é um arranjo de uma só dimensão que, portanto, necessita um só índice para acessar seus elementos.

Características de um vetor:

- nome, comum a todos os seus elementos;
- índices que identificam, de modo único, a posição dos elementos dentro do vetor;
- tipo dos elementos (inteiros, reais, etc.), comum a todos os elementos do vetor;
- conteúdo de cada elemento do vetor.

Na Figura 6.1 está representado graficamente um vetor de 10 elementos chamado de `nota`, com índices variando de 1 a 10, sendo o conteúdo do sétimo elemento igual a 8,5.

6.2.1 declaração de um vetor

Em pseudolinguagem, um arranjo é declarado com o seguinte tipo:

```
arranjo [<índice menor>..<índice maior>] de <tipo do vetor>
```

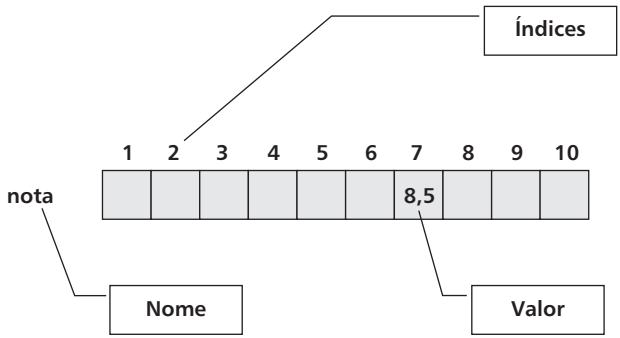

figura 6.1 Características de um vetor.

onde `índice menor` e `índice maior` definem os valores limites do intervalo de índices válidos para acessar o vetor, bem como o tipo de índice que deve ser usado, e `tipo do vetor` é o tipo dos dados que podem ser armazenados no vetor. Observar que o nome da variável vetor definida com esse tipo será comum a todos os seus elementos.

A declaração a seguir corresponde ao vetor da Figura 6.1, de nome `nota`, com 10 elementos reais acessados através dos índices de 1 a 10:

```
Variável: nota (arranjo [1..10] de real)
```

6.2.2 acesso a um elemento de um vetor

O **acesso a um elemento de um vetor** é feito indicando o nome do vetor seguido do índice do elemento considerado, colocado entre colchetes. Na pseudolinguagem utilizada, os índices válidos para acessar um vetor devem estar compreendidos entre os valores dos índices definidos na sua declaração, incluindo os extremos. Assim, a referência ao sétimo elemento do vetor `nota` é `nota[7]`.

Somente valores de índice válidos devem ser utilizados para acessar um vetor. Índices fora de intervalo provocam tentativas de acesso a áreas não previstas da memória, com resultados imprevisíveis e muitas vezes com reflexos aleatórios sobre o comportamento do algoritmo, gerando erros difíceis de localizar e corrigir. No caso do vetor `nota`, os índices válidos são os valores de 1 a 10.

Para identificar o índice de um elemento de um vetor podem ser utilizadas:

- uma constante, indicando diretamente o valor do índice. Assim, `nota[7]` referencia o sétimo elemento do vetor `nota`;
- uma variável, sendo que o valor contido nessa variável corresponde ao índice. Por exemplo, se, no momento do acesso, o valor da variável inteira `ind` for 3, então `nota[ind]` vai acessar o terceiro elemento do vetor `nota`;

- uma expressão que será avaliada, sendo seu resultado utilizado como índice. Por exemplo, se o valor da variável inteira `i` for 4, então `nota[i+1]` se refere ao quinto elemento do vetor `nota`.

Variáveis diferentes podem ser utilizadas como índice para acessar o mesmo vetor. No exemplo a seguir, o vetor `valor` é preenchido por leitura com o auxílio da variável `i` (primeiro comando `para/faça`) e, em seguida, seu conteúdo é exibido com o auxílio da variável `k` (segundo comando `para/faça`). Observar o uso da constante `MAX`, tanto para declarar o vetor quanto para acessá-lo.

```
Constante: MAX = 100
Variáveis:
   valor (arranjo [1..MAX] de inteiro)
   i, k (inteiro)
...
para i de 1 incr 1 até MAX faça
     ler (valor[i])
para k de 1 incr 1 até MAX faça
     escrever('Valor: ', valor[k])
```

A mesma variável pode ser utilizada, no mesmo momento ou em momentos diferentes, para acessar vetores diferentes. Por exemplo, supondo a declaração de dois vetores utilizados em um programa para corrigir uma prova, um contendo o gabarito das questões e o outro, as respostas de um aluno (Figura 6.2):

```
Constante: NUMQUEST = 30
Variáveis:
   gabarito, respostas (arranjo [1..NUMQUEST] de caractere)
```

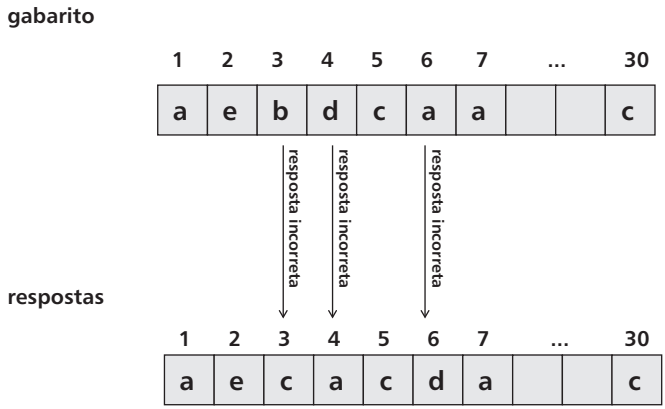

figura 6.2 Vetores `gabarito` e `respostas` (com respostas de um aluno).

O vetor `respostas` é reaproveitado a cada novo aluno processado. A correção da prova é feita comparando cada uma das respostas de um aluno com o elemento correspondente no vetor `gabarito`. O comando a seguir mostra a exibição dos valores desses dois vetores, o do gabarito e aquele com os resultados de um aluno, da primeira questão até a última, o que é feito utilizando o mesmo índice para acessar os dois vetores:

```
para k de 1 incr 1 até NUMQUEST faça
   escrever(gabarito[k], respostas[k])
```

6.2.3 inicialização de vetores

Não é necessário inicializar um vetor quando ele for totalmente preenchido por leitura, uma vez que todos os valores anteriormente armazenados nas posições de memória do vetor são descartados quando novos valores são colocados nelas.

Se um vetor for criado apenas parcialmente, restando posições não ocupadas no seu início, meio ou fim, ou se for utilizado em totalizações, devem ser tomados cuidados para garantir que os valores iniciais de todas as suas posições sejam os desejados. Isso pode ser feito ou inicializando o vetor na sua totalidade antes de sua utilização, ou inicializando cada posição do arranjo imediatamente antes de seu uso.

O trecho a seguir inicializa todos os elementos do vetor `nota`, de 10 elementos reais, com zeros:

```
para i de 1 incr 1 até 10 faça
   nota[i] ← 0
```

No caso de vetores preenchidos progressivamente, de forma contínua, visto que o número de posições ocupadas é sempre conhecido, é desnecessário inicializá-los antes do uso, desde que sempre sejam acessadas apenas as posições efetivamente ocupadas (ver Exercícios 6.2 e 6.3, a seguir).

6.3 → exemplos de uso de vetores

6.3.1 operações sobre um só vetor

Os trechos de código a seguir executam algumas operações sobre um vetor inteiro de 5 elementos chamado de `valor`:

```
Constante: MAX = 5
Variável: valor (arranjo [1..MAX] de inteiro)
```