Técnicas de Programação

TP0501

Prof. Giovane Barcelos
giovane_barcelos@uniritter.edu.br

Plano de Ensino
Conteúdo programático

1. Introdução à programação em C 2. Desenvolvimento estruturado de programas em C 3. Controle de programa 4. Funções 5. Arrays 6. Ponteiros	N1
7. Caracteres e strings 8. Entrada/Saída formatada 9. Estruturas, uniões, manipulações de bits e enumerações 10. Processamento de arquivos 11. Estruturas de dados 12. O pré-processador 13. Outros tópicos sobre C	N2

Objetivos

- A construir programas de forma modular a partir de pequenas partes chamadas funções.
- As funções matemáticas comuns na biblioteca-padrão em C.
- A criar novas funções.
- Os mecanismos usados para passar informações entre funções.
- Como o mecanismo de chamada/retorno de função é aceito pela pilha de chamada de função e pelos registros de ativação.
- Técnicas de simulação a partir da geração de números aleatórios.
- Como escrever e usar funções que chamam a si mesmas.

Objetivos

- A usar a estrutura de dados do array para representar listas e tabelas de valores.
- A definir um array, inicializá-lo e referir-se a seus elementos individuais.
- A definir constantes simbólicas.
- A passar arrays para funções.
- A usar arrays para armazenar, classificar e pesquisar listas e tabelas de valores.
- A definir e manipular arrays multidimensionais.
- Vai, pois, agora, escreve isto em uma tábua perante eles e registra-o em um livro.

Introdução

- Este capítulo serve como introdução ao importante assunto das estruturas de dados.
- **Arrays** são estruturas de dados que consistem em itens de dados relacionados do mesmo tipo.
- Adiante discutiremos a noção de struct (estrutura) — uma estrutura de dados que consiste em itens de dados relacionados, possivelmente de tipos diferentes.
- Arrays e estruturas são entidades 'estáticas' porque permanecem do mesmo tamanho ao longo de toda a execução do programa (elas podem, é claro, ser de uma classe de armazenamento automática e, portanto, ser criadas e destruídas sempre que os blocos em que estiverem definidas forem iniciados e finalizados).
- Em outro encontro apresentaremos estruturas de dados dinâmicas como listas, filas, pilhas e árvores, que podem crescer e encolher à medida que o programa é executado.

Arrays

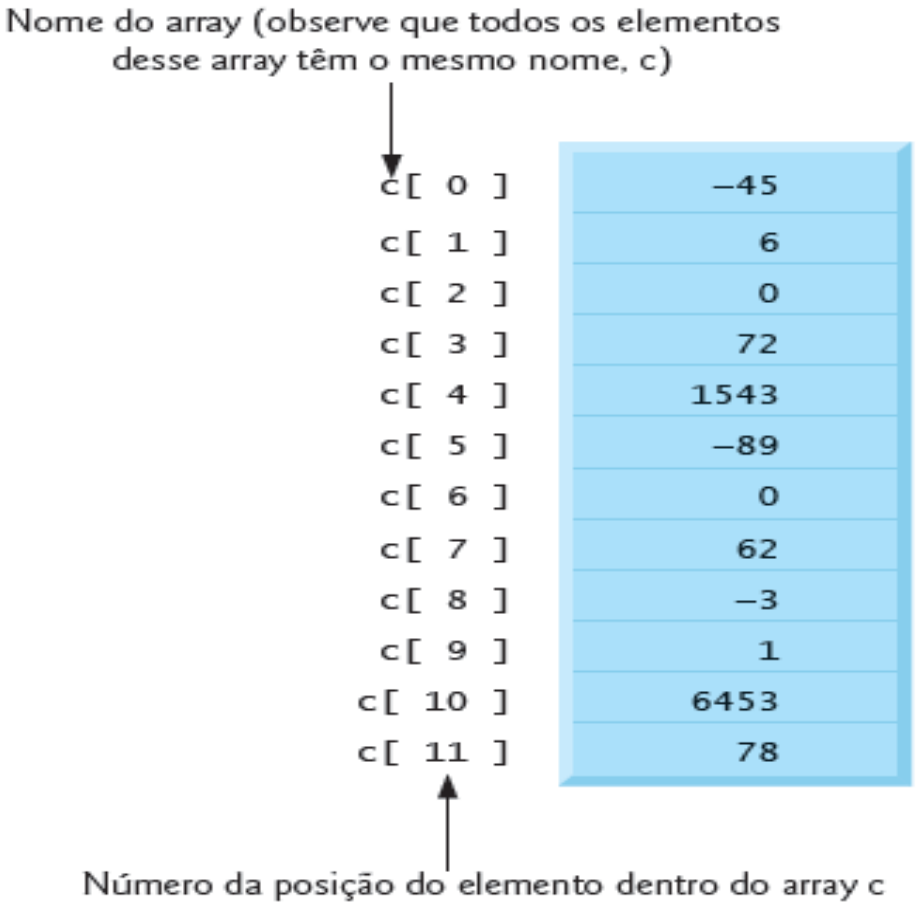


Figura 6.1 ■ Array de 12 elementos.

Arrays

- Um array é um conjunto de espaços de memória que se relacionam pelo fato de que todos têm o mesmo nome e o mesmo tipo.
- Para se referir a um local ou elemento em particular no array, especificamos o nome do array e o número da posição do elemento em particular no array.
- A Figura 6.1 mostra um array de inteiros chamado c.
- Esse array contém 12 elementos.
- Qualquer um desses elementos pode ser referenciado com o nome do array seguido pelo número de posição do elemento em particular entre colchetes ([ ]).

Arrays

- O primeiro elemento em cada array é identificado pela posição elemento **zerésimo**.
- Assim, o primeiro elemento do array c é referenciado como c[0], o segundo elemento, como c[1], o sétimo elemento, como c is referred to as c[6], e, em geral, o *i*-ésimo elemento do array c é referenciado como c[i - 1].
- Os nomes do array, como outros nomes de variável, só podem conter letras, dígitos e sublinhados.
- Nomes de array não podem começar por um dígito.
- O número da posição contido dentro dos colchetes é formalmente chamado de **subscrito** (ou **índice**).
- Um subscrito precisa ser um inteiro ou uma expressão inteira.

Arrays

- Se um programa usa uma expressão como um subscrito, então a expressão é avaliada para determinar o subscrito.
- Por exemplo, se a = 5 e b = 6, então a instrução
  - c[a + b] += 2;
- soma 2 ao elemento de array c[11].
- Um nome de array subscritado é um *lvalue* — ele pode ser usado no lado esquerdo de uma atribuição.

Arrays

- Examinemos o array c (Figura 6.1) mais de perto.
- O nome do array é c.
- Seus 12 elementos são referenciados como c[0], c[1], c[2], ..., c[11].
- O valor armazenado em c[0] é –45, o valor de c[1] é 6, o valor de c[2] é 0, o valor de c[7] é 62 e o valor de c[11] é 78.
- Para imprimir a soma dos valores contidos nos três primeiros elementos do array c, escreveríamos
  - printf("%d", c[0] + c[1] + c[2]);
- Para dividir o valor do sétimo elemento do array c por 2 e atribuir o resultado à variável x, escreveríamos

Erro comum de programação 6.1

É importante observar a diferença entre o 'sétimo elemento do array' e o 'elemento de array sete'. Como os subscritos de array começam em 0, o 'sétimo elemento do array' tem um subscrito 6, enquanto o 'elemento de array sete' tem um subscrito 7 e, na realidade, é o oitavo elemento do array. Esta é uma fonte de erros de 'diferença por um'.

Arrays

Operadores	Associatividade	Tipo
[ ] ( )	esquerda para direita	mais alta
++ -- ! (*tipo*)	direita para esquerda	unário
* / %	esquerda para direita	multiplicativo
+ -	esquerda para direita	aditivo
< <= > >=	esquerda para direita	relacional
== !=	esquerda para direita	igualdade
&&	esquerda para direita	AND lógico
||	esquerda para direita	OR lógico
?:	direita para esquerda	condicional
= += -= *= /= %=	direita para esquerda	atribuição
,	esquerda para direita	vírgula

Figura 6.2 ■ Precedência e associatividade de operadores.

Arrays

- Os colchetes usados para delimitar o subscrito de um array são realmente considerados como um operador em C.
- Eles têm o mesmo nível de precedência do operador de chamada de função (ou seja, os parênteses que são colocados após o nome da função para chamar essa função).
- A Figura 6.2 mostra a precedência e a associatividade dos operadores introduzidos até agora.

Declarando arrays

- Os arrays ocupam espaço na memória.
- Você especifica o tipo de cada elemento e o número de elementos exigidos por array de modo que o computador possa reservar a quantidade de memória apropriada.
- A declaração
  - `int c[ 12 ];`
- é usada para pedir ao computador que reserve 12 elementos para o array de inteiros c.

Declarando arrays

- A declaração a seguir
  - `int b[ 100 ], x[ 27 ];`

  reserva 100 elementos para o array de inteiros b ereserva 100 elementos para o array de inteiros x.
- Os arrays podem conter outros tipos de dados.
- As strings de caracteres e sua semelhança com os arrays serão discutidos nos próximos encontros
- A relação entre ponteiros e arrays será discutida quanto tratarmos sobre ponteiros

Exemplos de arrays

```c
/* Figura 6.3: fig06_03.c
   Inicializando um array */
#include <stdio.h>

/* função main inicia a execução do programa */
int main( void )
{
   int n[ 10 ]; /* n é um array de 10 inteiros */
   int i; /* contador */

   /* inicializa elementos do array n como 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = 0; /* define elemento no local i como 0 */
   } /* fim do for */

   printf( "%s%13s\n", "Elemento", "Valor" );

   /* saída na tela de conteúdo do array n em formato tabular */
   for ( i = 0; i < 10; i++ ) {
      printf( "%7d%13d\n", i, n[ i ] );
   } /* fim do for */

   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Elemento        Valor
       0            0
       1            0
       2            0
       3            0
       4            0
       5            0
       6            0
       7            0
       8            0
       9            0
```

Figura 6.3 ■ Inicializando os elementos de um array com zeros.

Exemplos de arrays

- A Figura 6.3 usa estruturas for para inicializar os elementos de um array n de 10 elementos do tipo inteiro com zeros e imprimir o array em formato tabular.
- A primeira instrução printf (linha 16) apresenta os cabeçalhos de coluna para as duas colunas impressas na estrutura for subsequente..

Exemplos de arrays

```c
/* Figura 6.4: fig06_04.c
   Inicializando um array com uma lista de inicializadores */
#include <stdio.h>

/* função main inicia a execução do programa */
int main( void )
{
   /* usa lista de inicializadores para inicializar array n */
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
   int i; /* contador */

   printf( "%s%13s\n", "Elemento", "Valor" );

   /* lista conteúdo do array em formato tabular */
   for ( i = 0; i < 10; i++ ) {
      printf( "%7d%13d\n", i, n[ i ] );
   } /* fim do for */

   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Elemento         Valor
       0            32
       1            27
       2            64
       3            18
       4            95
       5            14
       6            90
       7            70
       8            60
       9            37
```

Figura 6.4 ■ Inicializando os elementos de um array com uma lista de inicializadores.

Exemplos de arrays

- **Os elementos de um array também podem ser inicializados quando o array é declarado a partir de uma declaração com um sinal de igual e chaves, {}, que contenha uma lista de initializers separados por vírgula.**
- **A Figura 6.4 inicializa um array de inteiros com 10 valores (linha 9) e imprime o array em formato tabular.**

Exemplos de arrays

- Se houver menos inicializadores que elementos no array, os elementos restantes serão inicializados em zero.
- Por exemplo, os elementos do array n na Figura 6.3 poderiam ter sido inicializados com zero, da seguinte forma:
  - `int n[ 10 ] = { 0 };`
- Isso inicializa explicitamente o primeiro elemento em zero e inicializa os nove elementos restantes em zero, pois existem menos inicializadores que elementos no array.

Exemplos de arrays

- **É importante lembrar que os arrays não são automaticamente inicializados em zero.**
- **Você precisa, pelo menos, inicializar o primeiro elemento em zero para que os elementos restantes sejam automaticamente zerados.**
- **Esse método de inicialização dos elementos do array em 0 éposto em prática na compilação dos arrays static e no tempo de execução de arrays automáticos.**

Exemplos de arrays

Erro comum de programação 6.2

Esquecer-se de inicializar os elementos de um array cujos elementos deveriam ser inicializados.

Exemplos de arrays

- **A declaração de array**
  - `int n[ 5 ] = { 32, 27, 64, 18, 95, 14 };`

causa um erro de sintaxe porque existem seis inicializadores e apenas cinco elementos de array.

Exemplos de arrays

Erro comum de programação 6.3

Fornecer mais inicializadores em uma lista de inicializadores de array que a quantidade de elementos existentes no array é um erro de sintaxe.

Exemplos de arrays

- Se o tamanho do array for omitido de uma declaração com uma lista de inicializadores, o número de elementos no array será o número de elementos na lista de inicializadores.
- Por exemplo,
  - `int n[] = { 1, 2, 3, 4, 5 };`

criaria um array de cinco elementos.

Exemplos de arrays

```c
/* Figura 6.5: fig06_05.c
   Inicializa elementos do array s como inteiros pares de 2 a 20 */
#include <stdio.h>
#define SIZE 10 /* tamanho máximo do array */

/* função main inicia a execução do programa */
int main( void )
{
   /* constante simbólica SIZE pode ser usada para especificar tamanho do array */
   int s[ SIZE ]; /* array s tem SIZE elementos */
   int j; /* contador */

   for ( j = 0; j < SIZE; j++ ) { /* define os elementos */
      s[ j ] = 2 + 2 * j;
   } /* fim do for */

   printf( "%s%13s\n", "Elemento", "Valor" );

   /* lista de impressão do conteúdo do array s em formato tabular */
   for ( j = 0; j < SIZE; j++ ) {
      printf( "%7d%13d\n", j, s[ j ] );
   } /* fim do for */

   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Elemento        Valor
      0            2
      1            4
      2            6
      3            8
      4           10
      5           12
      6           14
      7           16
      8           18
      9           20
```

Figura 6.5 ■ Inicializando os elementos do array s com inteiros pares de 2 a 20.

Exemplos de arrays

- A Figura 6.5 inicializa os elementos de um array de 10 elementos s com os valores 2, 4, 6, ..., 20 imprime o array em formato tabular.
- Os valores são gerados ao se multiplicar o contador do loop por 2 e somar 2.

Exemplos de arrays

- A diretiva do pré-processador #define é introduzida nesse programa.
- A linha 4
  - **#define SIZE 10**

  define uma constante simbólica SIZE cujo valor é 10.
- Uma constante simbólica é um identificador substituído com o texto substituto pelo pré-processador C antes de o programa ser compilado.

Exemplos de arrays

- Quando o programa é pré-processado, todas as ocorrências da constante simbólica SIZE são substituídas com o texto substituto 10.
- O uso de constantes simbólicas para especificar tamanhos de array torna os programas mais escaláveis.
- Na Figura 6.5, poderíamos fazer o primeiro for (linha 13) preencher um array de 1000 elementos simplesmente mudando o valor de SIZE na diretiva #define de 10 para 1000.
- Se a constante simbólica SIZE não tivesse sido usada, teríamos de mudar o programa em três lugares separados para fazer com que ele tratasse de 1000 elementos de array.

Exemplos de arrays

Erro comum de programação 6.4

Terminar uma diretiva de pré-processador #define *ou* #include *com um ponto e vírgula. Lembre-se de que as diretivas de pré-processador não são comandos da linguagem C.*

Exemplos de arrays

- **Se, na linha 4, a diretiva de pré-processador #define terminar com um ponto e vírgula, todas as ocorrências da constante simbólica SIZE no programa serão substituídas com o texto 10; pelo pré-processador.**
- **Isso pode gerar erros de sintaxe durante a compilação ou erros lógicos durante a execução.**

Exemplos de arrays

Erro comum de programação 6.5

Atribuir um valor a uma constante simbólica em uma instrução executável é um erro de sintaxe. Uma constante simbólica não é uma variável. Nenhum espaço é reservado para ela pelo compilador, como ocorre nas variáveis, que mantêm valores em tempo de execução.

Observação sobre engenharia de software 6.1

Definir o tamanho de cada array como uma constante simbólica torna os programas mais escaláveis.

Exemplos de arrays

Boa prática de programação 6.1

Use apenas letras maiúsculas para nomes de constantes simbólicas. Isso faz com que essas constantes se destaquem
em um programa, além de lembrá-lo de que as constantes simbólicas não são variáveis.

Boa prática de programação 6.2

Nos nomes de constantes simbólicas com várias palavras, use sublinhados para separar as palavras e aumentar a legibilidade.

Exemplos de arrays

```
1  /* Figura 6.6: fig06_06.c
2     Calcula a soma dos elementos do array */
3  #include <stdio.h>
4  #define SIZE 12
5
6  /* função main inicia a execução do programa */
7  int main( void )
8  {
9     /* usa lista inicializadora para inicializar array */
10    int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
11    int i; /* contador */
12    int total = 0; /* soma do array */
13
14    /* conteúdo da soma do array a */
15    for ( i = 0; i < SIZE; i++ ) {
16       total += a[ i ];
17    } /* fim do for */
18
19    printf( "Total de valores dos elementos do array é %d\n", total );
20    return 0; /* indica conclusão bem-sucedida */
21 } /* fim do main */
```

```
Total de valores dos elementos do array é 383
```

Figura 6.6 ■ Cálculo da soma dos elementos de um array.

Exemplos de arrays

- A Figura 6.6 soma os valores contidos no array de inteiros de 12 elementos a.
- O corpo da estrutura for (linha 16) realiza o cálculo.

Exemplos de arrays

```
 1  /* Figura 6.7: fig06_07.c
 2     Programa de pesquisa com estudantes */
 3  #include <stdio.h>
 4  #define RESPONSE_SIZE 40 /* define tamanhos de array */
 5  #define FREQUENCY_SIZE 11
 6
 7  /* função main inicia a execução do programa */
 8  int main( void )
 9  {
10     int answer; /* contador para percorrer 40 respostas */
11     int rating; /* contador para percorrer frequências 1-10 */
12
13     /* inicializa contadores de frequência em 0 */
14     int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16     /* coloca as respostas da pesquisa no array responses */
17     int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
18        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
19        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
20
21     /* para cada resposta, seleciona valor de um elemento do array responses
22        e usa esse valor como subscrito na frequência do array para
23        determinar elemento a ser incrementado */
```

Exemplos de arrays

```
24      for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
25         ++frequency[ responses [ answer ] ];
26      } /* fim do for */
27
28      /* mostra resultados */
29      printf( "%s%17s\n", "Avaliação", "Frequência" );
30
31      /* listas de impressão das frequências em um formato tabular */
32      for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
33         printf( "%6d%17d\n", rating, frequency[ rating ] );
34      } /* fim do for */
35
36      return 0; /* indica conclusão bem-sucedida */
37   } /* fim do main */
```

```
Avaliação         Frequência
1                 2
2                 2
3                 2
4                 2
5                 5
6                 11
7                 5
8                 7
9                 1
10                3
```

Figura 6.7 ■ Programa de análise de pesquisa de alunos.

Exemplos de arrays

- Nosso próximo exemplo usa arrays para resumir os resultados dos dados coletados em uma pesquisa.
- Considere um problema com o seguinte enunciado.
  - Pedimos a 40 alunos que avaliassem a comida da cantina estudantil e dessem notas que fossem de 1 a 10 (1 significaria horrorosa e 10 significaria excelente). Coloque as 40 respostas em um array de inteiros e resuma os resultados da pesquisa.
- Esta é uma aplicação típica de arrays (ver Figura 6.7).
- Queremos resumir o número de respostas de cada tipo (ou seja, de 1 a 10).

Exemplos de arrays

- O array responses (linha 17) consiste em um array de 40 elementos com as respostas dos alunos.
- Usamos um array frequency (line 14) com 11 elementos (linha 14) para contar o número de ocorrências de cada resposta.
- Ignoramos frequency[0] porque é mais lógico fazer com que a resposta 1 incremente frequency[1] que frequency[0].
- Isso permite o uso direto de cada resposta como subscritas do array frequency.

Exemplos de arrays

Boa prática de programação 6.3
Empenhe-se em obter o máximo de clareza para o programa. Às vezes, vale a pena trocar o uso mais eficiente da
memória, ou o tempo do processador, por uma escrita de programas mais clara.

Dica de desempenho 6.1
Às vezes, considerações de desempenho superam considerações de clareza.

Exemplos de arrays

- O loop for (linha) obtém as respostas uma de cada vez do array responses e incrementa um dos 10 contadores (frequency[1] a frequency[10]) no array frequency.
- A instrução-chave do loop está na linha 25
  - ++frequency[responses[answer]];

  que incrementa o contador frequency apropriado, dependendo do valor de responses[answer].

Exemplos de arrays

- Quando a variável contadora answer é 0, responses[answer] é 1, de modo que ++frequency[responses[answer]]; é interpretado como
  - ++frequency[1];

  que incrementa o elemento de array um.
- Quando answer é 1, responses[answer] é 2, de modo que ++frequency[responses[answer]]; é interpretado como
  - ++frequency[2];

que incrementa o elemento de array dois.

Exemplos de arrays

- Quando answer é 2, responses[answer] é 6, de modo que ++frequency[responses[answer]]; é interpretado como
  - ++frequency[6];

  que incrementa o elemento de array seis e assim por diante.
- Independentemente do número de respostas processadas na pesquisa, somente um array de 11 elementos é necessário (ignorando o elemento zero) para resumir os resultados.
- Se os dados contivessem valores inválidos, como 13, o programa tentaria somar 1 a frequency[13].
- Isso estaria fora dos limites do array.

Exemplos de arrays

- ***C não ~~tem~~ faz a verificação ~~de~~ dos limites do array para impedir que o programa se refira a um elemento que não existe.***
- **Assim, um programa em execução pode ultrapassar o final de um array sem aviso.**
- **Você deverá garantir que todas as referências de array permaneçam dentro dos limites do array.**

Exemplos de arrays

Erro comum de programação 6.6

Referir-se a um elemento fora dos limites do array.

Dica de prevenção de erro 6.1

Ao realizar o percurso por um laço de repetição (looping) por um array, o subscrito do array nunca deverá ser
menor que 0, e sempre deverá ser menor que o número total de elementos no array (tamanho – 1). Cuide para que a condição de término do loop impeça o acesso a elementos fora desse intervalo.

Exemplos de arrays

Dica de prevenção de erro 6.2

Os programas deverão validar a exatidão de todos os valores de entrada, para impedir que informações errôneas
afetem os cálculos de um programa.

Exemplos de arrays

```
1  /* Figura 6.8: fig06_08.c
2     Programa de impressão de histograma */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* função main inicia a execução do programa */
7  int main( void )
8  {
9     /* usa lista de inicializadores para inicializar array n */
10    int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
11    int i; /* contador do for externo para elementos do array */
12    int j; /* contador do for interno conta *s em cada barra do histograma */
13
14    printf( "%s%13s%17s\n", "Elemento", "Valor", "Histograma" );
15
16    /* para cada elemento do array n, mostra uma barra do histograma */
17    for ( i = 0; i < SIZE; i++ ) {
```

Figura 6.8 ■ Impressão do histograma. (Parte 1 de 2.)

Exemplos de arrays

```
18          printf( "%7d%13d", i, n[ i ]) ;
19
20          for ( j = 1; j <= n[ i ]; j++ ) { /* imprime uma barra */
21             printf( "%c", '*' );
22          } /* fim do for interno */
23
24          printf( "\n" ); /* fim de uma barra do histograma */
25       } /* fim do for externo */
26
27       return 0; /* indica conclusão bem-sucedida */
28    } /* fim do main */
```

```
Elemento        Valor           Histograma
       0           19           ********************
       1            3           ***
       2           15           ***************
       3            7           *******
       4           11           ***********
       5            9           *********
       6           13           *************
       7            5           *****
       8           17           *****************
       9            1           *
```

Figura 6.8 ■ Impressão do histograma. (Parte 2 de 2.)

Exemplos de arrays

- Nosso próximo exemplo (Figura 6.8) lê os números de um array e apresenta a informação graficamente na forma de um gráfico de barras, ou histograma — cada número é impresso, e, depois, uma barra que consiste na quantidade de asteriscos correspondente é impressa ao lado do número.
- A estrutura for aninhada (linha 20) desenha as barras.
- Observe o uso de printf("\n") para encerrar uma barra do histograma (linha 24).

Exemplos de arrays

```
1  /* Figura 6.9: fig06_09.c
2     Lança um dado de 6 lados 6000 vezes */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #define SIZE 7
7
8  /* função main inicia a execução do programa */
9  int main( void )
10    {
11    int face; /* valor aleatório de 1 - 6 do dado */
12    int roll; /* contador de lançamentos de 1-6000 */
13    int frequency[ SIZE ] = { 0 }; /* limpa contadores */
14
15    srand( time( NULL ) ); /* semente do gerador de números aleatórios */
16
17    /* rola dado 6000 vezes */
18    for ( roll = 1; roll <= 6000; roll++ ) {
19       face = 1 + rand() % 6;
20       ++frequency[ face ]; /* substitui switch de 26 linhas da Fig. 5.8 */
21    } /* fim do for */
22
23    printf( "%s%17s\n", "Face", "Frequency" );
24
25    /* mostra elementos de frequência 1-6 em formato tabular */
```

Figura 6.9 ■ Programa de lançamento de dados que usa um array no lugar de switch. (Parte 1 de 2.)

Exemplos de arrays

```
26      for ( face = 1; face < SIZE; face++ ) {
27         printf( "%4d%17d\n", face, frequency[ face ] );
28      } /* fim do for */
29
30      return 0; /* indica conclusão bem-sucedida */
31   } /* fim do main */
```

```
Face            Frequência
1               1029
2               951
3               987
4               1033
5               1010
6               990
```

Figura 6.9 ■ Programa de lançamento de dados que usa um array no lugar de switch. (Parte 2 de 2.)

Exemplos de arrays

- **O problema era lançar um único dado de seis lados 6000 vezes para testar se o gerador de números aleatórios realmente produziria números aleatórios.**
- **Uma versão de array desse programa aparece na Figura 6.9.**

Exemplos de arrays

- Até aqui, a única capacidade de processamento de strings que temos é a de exibir uma string com printf.
- Uma string como "hello" é, na realidade, um array static de caracteres individuais em C.
- Um array de caracteres pode ser inicializado com uma string literal.
- Por exemplo,
  - `char string1[] = "first";`

  inicializa os elementos do array string1 com os caracteres individuais na string literal "first".

Exemplos de arrays

- Nesse caso, o tamanho do array string1 é determinado pelo compilador com base no comprimento da string.
- A string "first" contém cinco caracteres e *mais* um caractere de término de string, chamado caractere nulo.
- Assim, o array string1, na realidade, contém seis elementos.
- A constante de caractere que representa o caractere nulo é'\0'.
- Todas as strings em C terminam com esse caractere.
- Um array de caracteres que represente uma string sempre deverá ser definido com tamanho suficiente para manter o número de caracteres na string e o caractere nulo de finalização.
- Os arrays de caracteres também podem ser inicializados com constantes de caractere individuais em uma lista de inicializadores.

Exemplos de arrays

- A definição anterior é equivalente a
  - `char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };`
- Como uma string é, na realidade, um array de caracteres, podemos acessar diretamente os caracteres individuais em uma string usando a notação de subscrito de array.
- Por exemplo, string1[0] é o caractere'f' e string1[3] é o caractere's'.
- Também podemos entrar com uma string diretamente em um array de caracteres a partir do teclado, usando scanf o especificador de conversão %s.

Exemplos de arrays

- Por exemplo,
  - `char string2[ 20 ];`

  cria um array de caracteres capaz de armazenar uma string de no máximo 19 caracteres e um caractere nulo de finalização.
- A instrução
  - `scanf( "%s", string2 );`

  lê uma string do teclado para string2.
- O nome do array é passado para scanf sem o & anterior, usado com variáveis não string.
- O & normalmente é utilizado para dar a scanf o local de uma variável na memória, para que um valor possa ser armazenado ali.

Exemplos de arrays

- Na Seção 6.5, discutiremos a passagem de arrays a funções, e veremos que o valor de um nome de array é o endereço do início do array; portanto, o & não é necessário.
- A função scanf lerá caracteres até encontrar um espaço, uma tabulação, uma nova linha ou um indicador de fim de arquivo.
- A string não deverá ter mais de 19 caracteres, para que possa deixar espaço para o caractere nulo de finalizaçao.
- Se o usuário digitar 20 ou mais caracteres, seu programa falhará! Por esse motivo, use o especificador de formato %19s, para ƀ que scanf não escreva caracteres na memória além do final do array s.

Exemplos de arrays

- **Você é responsável por garantir que o array em que a string será lida seja capaz de manter qualquer string que o usuário digitar no teclado.**
- **A função scanf lê caracteres do teclado até que o primeiro caractere de espaço em branco seja encontrado; ele não verifica o tamanho do array.**
- **Assim, scanf pode continuar escrevendo mesmo depois de o array terminar.**

Exemplos de arrays

Erro comum de programação 6.7

Não fornecer a scanf *com um array de caracteres grande o suficiente para armazenar uma string digitada no*
teclado pode resultar em destruição de dados em um programa e outros erros em tempo de execução. Isso também pode tornar o sistema suscetível a ataques de worm e vírus.

Exemplos de arrays

- Um array de caracteres que represente uma string pode ser exibido com printf e o especificador de conversão %s.
- O array string2 é exibido com a instrução
  - printf("%s\n", string2);
- A função printf, assim como scanf, não verifica o tamanho do array de caracteres.
- Os caracteres da string são impressos até que seja encontrado um caractere nulo de finalização.

Exemplos de arrays

```c
/* Figura 6.10: fig06_10.c
   Tratando arrays de caracteres como strings */
#include <stdio.h>

/* função main inicia a execução do programa */
int main( void )
{
   char string1[ 20 ]; /* reserva 20 caracteres */
   char string2[] = "string literal"; /* reserva 15 caracteres */
   int i; /* contador */

   /* lê substring do usuário para array string1 */
   printf("Digite uma string: ");
   scanf( "%s", string1 ); /* entrada terminada com espaço em branco */

   /* mostra strings */
   printf( "string1 é: %s\nstring2 is: %s\n"
           "string1 com espaços entre caracteres é:\n",
           string1, string2 );

   /* mostra caracteres até o caractere nulo ser alcançado */
   for ( i = 0; string1[ i ] != '\0'; i++ ) {
      printf( "%c ", string1[ i ] );
   } /* fim do for */

   printf( "\n" );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Digite uma string: Olá
string1 é: Olá
string2 é: string literal
string1 com espaços entre caracteres é:
O l á
```

Figura 6.10 ■ Tratando arrays de caracteres como strings.

Exemplos de arrays

- **A Figura 6.10 demonstra a inicialização de um array de caracteres com uma string literal, com a leitura de uma string em um array de caracteres, com a exibição de um array de caracteres como uma string e com o acesso de caracteres individuais de uma string.**

Exemplos de arrays

- **A Figura 6.10 usa uma estrutura for (linha 22) para percorrer do array string1 e imprimir os caracteres individuais separados por espaços, usando o especificador de conversão %c.**
- **A condição para a estrutura for statement, string1[i] != '\0', é verdadeira enquanto o caractere nulo de finalização não tiver sido encontrado na string..**

Exemplos de arrays

- **Uma variável local static existe enquanto durar o programa, mas é visível apenas no corpo da função.**
- **Podemos aplicar static a uma definição de array local para que o array não seja criado e inicializado toda vez que a função for chamada, e o array não seja destruído toda vez que a função for concluída no programa.**
- **Isso reduz o tempo de execução do programa, principalmente no caso de programas que contenham arrays grandes e tenham funções usadas com frequência.**

Exemplos de arrays

Dica de desempenho 6.2
Nas funções que contêm arrays automáticos, em que a função entra e sai do escopo com frequência, torne o array
static *para que ele não seja criado toda vez que a função for chamada.*

Exemplos de arrays

- Arrays static são inicializados uma vez, em tempo de compilação.
- Se você não inicializar(sugiro trocar esta frase deveria ser impessoal) explicitamente um array static, os elementos desse array serão inicializados ~~em~~ zero pelo compilador.
- A Figura 6.11 demonstra a função staticArrayInit (linha 22) com um array local static (linha 25), e a função automaticArrayInit (linha 44) com um array automático local (linha 47).
- A função staticArrayInit é chamada duas vezes (linhas 12 e 16).
- O array static local, na função, é inicializado em zero pelo compilador (linha 25).
- A função exibe o array, adiciona 5 em cada elemento e exibe o array novamente.

Exemplos de arrays

- Na segunda vez que a função é chamada, o array static contém os valores armazenados durante a primeira chamada de função.
- A função automaticArrayInit também é chamada duas vezes (linhas 13 e 17).
- Os elementos do array local automático na função são inicializados com os valores 1, 2 e 3 (linha 47).
- A função exibe o array, adiciona 5 em cada elemento e exibe o array novamente.
- Na segunda vez que a função é chamada, os elementos do array são inicializados em 1, 2 e 3 novamente, pois o array tem duração de armazenamento automática.

Exemplos de arrays

Erro comum de programação 6.8

Pressupor que os elementos de um array local static *sejam inicializados em zero toda vez que a função em que o array é definido for chamada.*

Exemplos de arrays

```
1  /* Figura 6.11: fig06_11.c
2     Arrays estáticos são inicializados em zero */
3  #include <stdio.h>
4
5  void staticArrayInit( void ); /* protótipo de função */
6  void automaticArrayInit( void ); /* protótipo de função */
7
8  /* função main inicia a execução do programa */
9  int main( void )
10 {
11    printf( "Primeira chamada para cada função:\n" );
12    staticArrayInit();
13    automaticArrayInit();
14
15    printf( "\n\nSegunda chamada para cada função:\n" );
16    staticArrayInit();
17    automaticArrayInit();
18    return 0; /* indica conclusão bem-sucedida */
19 } /* fim do main */
20
```

Figura 6.11 ■ Arrays estáticos são inicializados em zero se não forem inicializados explicitamente. (Parte 1 de 2.)

Exemplos de arrays

```
21  /* função para demonstrar um array local estático */
22  void staticArrayInit( void )
23  {
24     /* inicializa elementos em 0 na primeira vez que a função é chamada */
25     static int array1[ 3 ];
26     int i; /* contador */
27
28     printf( "\nValores na entrada de staticArrayInit:\n" );
29
30     /* mostra conteúdo de array1 */
31     for ( i = 0; i <= 2; i++ ) {
32        printf( "array1[ %d ] = %d  ", i, array1[ i ] );
33     } /* fim do for */
34
35     printf( "\nValores na saída de staticArrayInit:\n" );
36
37     /* modifica e mostra conteúdo de array1 */
38     for ( i = 0; i <= 2; i++ ) {
39        printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
40     } /* fim do for */
41  } /* fim da função staticArrayInit */
42
43  /* função para demonstrar um array lógico automático */
44  void automaticArrayInit( void )
45  {
46     /* inicializa elementos toda vez que a função é chamada */
47     int array2[ 3 ] = { 1, 2, 3 };
48     int i; /* contador */
49
50     printf( "\n\nValores na entrada de automaticArrayInit:\n" );
51
52     /* exibe conteúdo de array2 */
53     for ( i = 0; i <= 2; i++ ) {
54        printf("array2[ %d ] = %d  ", i, array2[ i ] );
55     } /* fim do for */
56
```

Exemplos de arrays

```
57      printf( "\nValores na saída de automaticArrayInit:\n" );
58
59      /* modifica e exibe conteúdo de array2 */
60      for ( i = 0; i <= 2; i++ ) {
61         printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
62      } /* fim do for */
63   } /* fim da função automaticArrayInit */
```

```
Primeira chamada para cada função:
Valores na entrada de staticArrayInit:
array1[ 0 ] = 0 array1[ 1 ] = 0 array1[ 2 ] = 0
Valores na saída de staticArrayInit:
array1[ 0 ] = 5 array1[ 1 ] = 5 array1[ 2 ] = 5
Valores na entrada de automaticArrayInit:
array2[ 0 ] = 1 array2[ 1 ] = 2 array2[ 2 ] = 3
Valores na saída de automaticArrayInit:
array2[ 0 ] = 6 array2[ 1 ] = 7 array2[ 2 ] = 8
Segunda chamada para cada função:
Valores na entrada de staticArrayInit:
array1[ 0 ] = 5 array1[ 1 ] = 5 array1[ 2 ] = 5
Valores na saída de staticArrayInit:
array1[ 0 ] = 10 array1[ 1 ] = 10 array1[ 2 ] = 10
Valores na entrada de automaticArrayInit:
array2[ 0 ] = 1 array2[ 1 ] = 2 array2[ 2 ] = 3
Valores na saída de automaticArrayInit:
array2[ 0 ] = 6 array2[ 1 ] = 7 array2[ 2 ] = 8
```

Figura 6.11 ■ Arrays estáticos são inicializados em zero se não forem inicializados explicitamente. (Parte 2 de 2.)

Passando arrays para funções

- Para passar um argumento de array a uma função, especifique o nome do array sem qualquer colchete.
- Por exemplo, se o array hourlyTemperatures tiver sido definido como
  - `int tempPorHora [ 24 ];`

  a chamada de função
  - `modifyArray(tempPorHora, 24 )`

  passa o array tempPorHora e seu tamanho à função modifyArray.

Passando arrays para funções

- Diferente de arrays char que contêm strings, outros tipos de array não possuem um valor finalizador especial. Por esse motivo, o tamanho de um array é passado à função, de modo que a função possa processar o número apropriado de elementos.
- C passa arrays a funções por referência automaticamente — as funções chamadas podem modificar os valores de elemento nos arrays originais das funções que os utilizam.
- O nome do array é avaliado como o endereço do primeiro elemento do array.
- Visto que o endereço inicial do array é passado, a função chamada sabe exatamente onde o array está armazenado.

Passando arrays para funções

```c
/* Figura 6.12: fig06_12.c
   O nome do array é o mesmo que o endereço de &array[ 0 ] */
#include <stdio.h>

/* função main inicia a execução do programa */
int main( void )
{
   char array[ 5 ]; /* define um array de tamanho 5 */

   printf( "  array = %p\n&array[0] = %p\n   &array = %p\n",
      array, &array[ 0 ], &array );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
array = 0012FF78
&array[0] = 0012FF78
&array = 0012FF78
```

Figura 6.12 ■ O nome do array é o mesmo que o endereço do primeiro elemento do array.

Passando arrays para funções

- **Portanto, quando a função chamada modifica os elementos do array no corpo de sua função, ela está modificando os elementos reais do array em seus locais de memória originais.**
- **A Figura 6.12 demonstra que um nome de array é, na realidade, o endereço do primeiro elemento de um array, exibindo array, &array[0] e &array usando o especificador de conversão %p — um especificador de conversão especial para imprimir endereços.**
- **O especificador de conversão %p normalmente exibe endereços como números hexadecimais.**

Passando arrays para funções

- Os números hexadecimais (base 16) consistem nos dígitos de 0 a 9 e nas letras de A até F (essas letras são os equivalentes hexadecimais dos números de 10 a 15).
- Elas normalmente são usadas como notação abreviada para grandes valores inteiros.
- O Apêndice C, Sistemas de Numeração, oferece uma discussão detalhada dos relacionamentos entre inteiros binários (base 2), octais (base 8), decimais (base 10; inteiros-padrão) e hexadecimais.
- A saída mostra que tanto array e &array[0] têm o mesmo valor, a saber, 0012FF78.

Passando arrays para funções

Dica de desempenho 6.3

Passar arrays por referência faz sentido por motivo de desempenho. Se os arrays fossem passados por valor, uma cópia de cada elemento seria passada. Para arrays grandes e passados com frequência, isso seria demorado, e consumiria espaço de armazenamento para as cópias dos arrays.

Observação sobre engenharia de software 6.2

É possível passar um array por valor (usando um truque simples que explicaremos no Capítulo 10).

Passando arrays para funções

- **Embora arrays inteiros sejam passados por referência, elementos individuais do array são passados por valor, exatamente como as variáveis simples.**
- **Esses pedaços de dados simples isolados (como ints, floats e chars) são chamados de escalares.**
- **Para passar um elemento de um array para uma função, use o nome subscritado do elemento de array como um argumento na chamada de função.**
- **Nos próximos encontros mostraremos como passar escalares (ou seja, variáveis e elementos individuais de array) para funções por referência.**

Passando arrays para funções

- Para uma função receber um array por meio de uma chamada de função, a lista de parâmetros da função precisa mencionar que um array será recebido.
- Por exemplo, o cabeçalho da função modifyArray (que mencionamos anteriormente nessa seção) poderia
- ser escrito como
  - `void modifyArray( int b[], int size )`

  indicando que modifyArray espera receber um array de inteiros no parâmetro b e o número de elementos de array no parâmetro size.
- O tamanho do array não precisa estar entre os colchetes.

Passando arrays para funções

```
1  /* Figura 6.13: fig06_13.c
2     Passando arrays e elementos individuais do array para funções */
3  #include <stdio.h>
4  #define SIZE 5
5
6  /* protótipos de função */
7  void modifyArray( int b[], int size );
8  void modifyElement( int e );
9
10 /* função main inicia a execução do programa */
11 int main( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* inicializa a */
14    int i; /* contador */
15
16    printf( "Efeitos da passagem do array inteiro por referência:\n\nOs "
17       "valores o array original são:\n" );
18
19    /* imprime na tela array original */
20    for ( i = 0; i < SIZE; i++ ) {
21       printf( "%3d", a[ i ] );
22    } /* fim do for */
23
24    printf( "\n" );
25
26    /* passa array a um modifyArray por referência */
27    modifyArray( a, SIZE );
```

Figura 6.13 ■ Passagem de arrays e de elementos individuais do array para funções. (Parte 1 de 2.)

Passando arrays para funções

```
28
29     printf( "Os valores do array modificado são:\n" );
30
31     /* array modificado na saída */
32     for ( i = 0; i < SIZE; i++ ) {
33        printf( "%3d", a[ i ] );
34     } /* fim do for */
35
36     /* valor de saída de a[ 3 ] */
37     printf( "\n\n\nEfeitos de passar elemento do array "
38        "por valor:\n\nO valor de a[3] é %d\n", a[ 3 ] );
39
40     modifyElement( a[ 3 ] ); /* passa elemento do array a[ 3 ] por valor */
41
42     /* mostra valor de a[ 3 ] */
43     printf( "O valor de a[ 3 ] é %d\n", a[ 3 ] );
44     return 0; /* indica conclusão bem-sucedida */
45  } /* fim do main */
46
47  /* na função modifyArray, "b" aponta para o array original "a"
48     na memória */
49  void modifyArray( int b[], int size )
50  {
51     int j; /* contador */
52
53     /* multiplica cada elemento do array por 2 */
54     for ( j = 0; j < size; j++ ) {
55        b[ j ] *= 2;
56     } /* fim do for */
57  } /* fim da função modifyArray */
58
59  /* na função modifyElement, "e" é uma cópia local do elemento
60     do array a[ 3 ] passado de main */
61  void modifyElement( int e )
62  {
63     /* multiplica parâmetro por 2 */
64     printf( "Valor em modifyElement é %d\n", e *= 2 );
65  } /* fim da função modifyElement */
```

Passando arrays para funções

```
Efeitos da passagem do array inteiro por referência:

Os valores do array original são:
 0 1 2 3 4
Os valores do array modificado são:
 0 2 4 6 8

Efeitos da passagem do elemento do array por valor:
O valor de a[3] é 6
Valor em modifyElement é 12
O valor de a[ 3 ] é 6
```

Figura 6.13 ■ Passagem de arrays e de elementos individuais do array para funções. (Parte 2 de 2.)

Passando arrays para funções

- Se ele for incluído, o compilador verificará se ele é maior que zero, e depois o excluirá.
- Especificar um tamanho negativo é um erro de compilação.
- Visto que os arrays são automaticamente passados por referência, quando a função chamada usa o nome de array b, ela estará referenciando o array na função chamadora (array hourlyTemperatures in the na chamada anterior).
- Figura 6.13 demonstra a diferença entre passar um array inteiro e passar um elemento do array.
- Primeiro, o programa exibe os cinco elementos do array de inteiros a (linhas 20-22).

Passando arrays para funções

- Em seguida, a e seu tamanho são passados para a função modifyArray (linha 27), onde cada um dos elementos de a é multiplicado por 2 (linhas 54-55).
- Depois, a é reimpresso em main (linhas 32–34).
- Como vemos na saída, os elementos de a são realmente modificados por modifyArray.
- Agora, o programa exibe o valor de a[3] (linha 38)
- e o passa para a função modifyElement (linha 40).
- A função modifyElement multiplica seu argumento por 2 (linha 64) e exibe o novo valor.
- a[3] não é modificado ao ser reimpresso em main (linha 43), pois os elementos individuais do array são passados por valor.

Passando arrays para funções

- Em seus programas, haverá situações em que uma função não deverá ter permissão para modificar elementos do array.
- Como os arrays são sempre passados por referência, a modificação dos valores em um array é difícil de controlar. C oferece o qualificador de tipo **const** para impedir a modificação dos valores do array em uma função.
- Quando um parâmetro de array é precedido pelo qualificador const os elementos do array se tornam constantes no corpo da função, e qualquer tentativa de modificar um elemento do array no corpo da função resulta em um erro no tempo de compilação. Isso permite que você corrija um programa para que ele não tente modificar os elementos do array.

Passando arrays para funções

```c
/* Figura 6.14: fig06_14.c
   Demonstrando o qualificador de tipo const com arrays */
#include <stdio.h>

void tryToModifyArray( const int b[] ); /* protótipo de função */

/* função main inicia a execução do programa */
int main( void )
{
   int a[] = { 10, 20, 30 }; /* inicializa a */

   tryToModifyArray( a );

   printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */

/* na função tryToModifyArray, array b é const, de modo que não pode ser
   usado para modificar o array original a em main. */
void tryToModifyArray( const int b[] )
{
   b[ 0 ] /= 2; /* erro */
   b[ 1 ] /= 2; /* erro */
   b[ 2 ] /= 2; /* erro */
} /* fim da função tryToModifyArray */
```

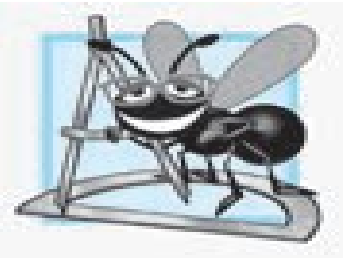
```
Compiling...
FIG06_14.C
fig06_14.c(22) : error C2166: l-value specifies const object
fig06_14.c(23) : error C2166: l-value specifies const object
fig06_14.c(24) : error C2166: l-value specifies const object
```

Figura 6.14 ■ Qualificador de tipo const.

Passando arrays para funções

- A Figura 6.14 demonstra o qualificador const.
- A função tryToModifyArray (linha 20) é definida com o parâmetro const int b[], que especifica que o array b é constante e não pode ser modificado.
- A saída mostra as mensagens de erro produzidas pelo compilador — os erros podem ser diferentes no seu sistema.
- Cada uma das três tentativas da função de modificar os elementos do array resulta no erro do compilador “l-value specifies a const object.” .

Passando arrays para funções



Observação sobre engenharia de software 6.3

O qualificador de tipo const *pode ser aplicado a um parâmetro de array em uma definição de função para impedir que o array original seja modificado no corpo da função. Este é outro exemplo do princípio do menor privilégio. As funções não devem ter a capacidade de modificar um array, a menos que seja absolutamente necessário.*

Ordenando Arrays

- A ordenação de dados (ou seja, a classificação dos dados em uma ordem em particular, crescente ou decrescente) é uma das aplicações mais importantes da computação.
- Neste capítulo, discutimos aquele que talvez seja o esquema de ordenação mais simples que se conheça.
- No Capítulo 12 e no Apêndice F, investigaremos esquemas mais complexos, que geram um desempenho superior.

Ordenando Arrays

Dica de desempenho 6.4

Normalmente, os algoritmos mais simples oferecem o pior desempenho. Sua virtude é que eles são fáceis de escrever, testar
e depurar. Algoritmos mais complexos normalmente são necessários para que se obtenha o máximo de desempenho.

Ordenando Arrays

```
 1  /* Figura 6.15: fig06_15.c
 2     Esse programa ordena os valores de um array em ordem crescente */
 3  #include <stdio.h>
 4  #define SIZE 10
 5
 6  /* função main inicia a execução do programa */
 7  int main( void )
 8  {
 9     /* inicializa a */
10     int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
11     int pass; /* contador de passada */
12     int i; /* contador de comparação */
13     int hold; /* local temporário usado para trocar elementos do array */
14
15     printf( "Itens de dados na ordem original\n" );
16
17     /* imprime array original */
18     for ( i = 0; i < SIZE; i++ ) {
19        printf( "%4d", a[ i ] );
20     } /* fim do for */
21
```

Ordenando Arrays

```
22      /* bubble sort */
23      /* loop para controlar número de passadas */
24      for ( pass = 1; pass < SIZE; pass++ ) {
25
26         /* loop para controlar número de comparações por passada */
27         for ( i = 0; i < SIZE - 1; i++ ) {
28
29            /* compara elementos adjacentes e os troca se o primeiro
30            elemento for maior que o segundo elemento */
31            if ( a[ i ] > a[ i + 1 ] ) {
32               hold = a[ i ];
33               a[ i ] = a[ i + 1 ];
34               a[ i + 1 ] = hold;
35            } /* fim do if */
36         } /* fim do for interno */
37      } /* fim do for externo */
38
39      printf( "\nItens de dados em ordem crescente\n" );
40
41      /* imprime array ordenado */
42      for ( i = 0; i < SIZE; i++ ) {
43         printf( "%4d", a[ i ] );
44      } /* fim do for */
45
46      printf( "\n" );
47      return 0; /* indica conclusão bem-sucedida */
48   } /* fim do main */
```

```
Itens de dados na ordem original
   2   6   4   8   10   12   89   68   45   37
Itens de dados em ordem crescente
   2   4   6   8   10   12   37   45   68   89
```

Figura 6.15 ■ Ordenando um array com o bubble sort.

Ordenando Arrays

- A Figura 6.15 ordena os valores nos elementos do array de dez elementos a (linha 10) em ordem crescente.
- A técnica que usamos é chamada de bubble sort ou sinking sort, pois os valores menores gradualmente sobem como 'bolhas' até o topo do array, como bolhas de ar que sobem para a superfície da água, enquanto os valores maiores vão para o fundo do array.
- A técnica é fazer várias passadas pelo array.
- A cada passada, pares sucessivos de elementos são comparados.
- Se um par está em ordem crescente (ou se os valores forem idênticos), deixamos os valores como estão. Se um par está em ordem decrescente, seus valores são trocados no array.

Ordenando Arrays

- Em primeiro lugar, o programa compara a[0] com a[1], depois a[1] com a[2], depois a[2] com a[3], e assim por diante, até completar a passada comparando a[8] com a[9].
- Embora existam 10 elementos, somente nove comparações são realizadas.
- Devido ao modo como as comparações sucessivas são feitas, um valor grande pode descer muitas posições no array em uma única passada, mas um valor pequeno pode subir apenas uma posição.
- Na primeira passada, o maior valor certamente afundará para o elemento do fundo do array, a[9].

Ordenando Arrays

- Na segunda passada, o segundo maior valor certamente afundará para a[8].
- Na nona passada, o nono maior valor afundará para a[1].
- Isso coloca o menor valor em a[0], de modo que apenas nove passadas do array serão necessárias para classificar o array, embora existam dez elementos
- A ordenação é realizada pelo loop for aninhado (linhas 24-37).

Ordenando Arrays

- Se uma troca for necessária, ela será realizada pelas três atribuições
  - hold = a[i];
    a[i] = a[i + 1];
    a[i + 1] = hold;

  em que a variável extra hold armazena temporariamente um dos dois valores que estão sendo trocados.
- A troca não pode ser realizada somente com as duas atribuições
  - a[i] = a[i + 1];
    a[i + 1] = a[i];

Ordenando Arrays

- Se, por exemplo, a[i] é e a[i7 + 1] é 5, após a primeira atribuição, os dois valores serão 5 e o valor 7 será perdido.
- Daí a necessidade da variável extra hold.
- A principal virtude do bubble sort é que ele é fácil de programar.
- Porém, o bubble sort é muito lento, pois cada troca move o elemento apenas uma posição mais próxima de seu destino final.
- Isso se torna aparente quando se ordena arrays grandes.
- Na seção de exercícios, iremos desenvolver versões mais eficientes do bubble sort.
- Ordenações muito mais eficientes do que o bubble sort vêm sendo desenvolvidas.
- Investigaremos algumas delas na seção de exercícios.

Ordenando Arrays

```c
/* Figura 6.16: fig06_16.c
   Esse programa introduz o tópico da análise de dados de pesquisa.
   Ele calcula a média, a mediana e a moda dos dados */
#include <stdio.h>
#define SIZE 99

/* protótipo de funções */
void mean( const int answer[] );
void median( int answer[] );
void mode( int freq[], const int answer[] ) ;
void bubbleSort( int a[] );
void printArray( const int a[] );

/* função main inicia a execução do programa */
int main( void )
{
   int frequency[ 10 ] = { 0 }; /* inicializa frequência do array */

   /* inicializa resposta do array */
   int response[ SIZE ] =
      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
        4, 5, 6, 1, 6, 5, 7, 8, 7 };

   /* processa respostas */
   mean( response );
   median( response );
   mode( frequency, response );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */

/* calcula média de todos os valores de resposta */
void mean( const int answer[] )
{
```

Ordenando Arrays

```
42     int j; /* contador para totalizar os elementos do array */
43     int total = 0; /* variável para manter a soma dos elementos do array */
44
45     printf( "%s\n%s\n%s\n", "********", "  Média", "********" );
46
47     /* valores totais de respostas */
48     for ( j = 0; j < SIZE; j++ ) {
49        total += answer[ j ];
50     } /* fim do for */
51
52     printf( "A média é o valor médio dos itens de dados.\n"
53             "A média é igual ao total de todos\n"
54             "os itens de dados divididos pelo número\n"
55             "de itens de dados ( %d ). O valor médio para esta\n"
56             "execução é: %d / %d = %.4f\n\n",
57             SIZE, total, SIZE, ( double ) total / SIZE );
58  } /* fim da função mean */
59
60  /* ordena array e determina valor do elemento mediano */
61  void median( int answer[] )
62  {
63     printf( "\n%s\n%s\n%s\n%s",
64             "********", " Mediana", "********",
65             "O array de respostas, não ordenado, é" );
66
```

Figura 6.16 ■ Programa de análise de dados para pesquisa. (Parte 1 de 3.)

Ordenando Arrays

```
67      printArray( answer ); /* exibe array não ordenado */
68
69      bubbleSort( answer ); /* ordena array */
70
71      printf( "\n\nO array ordenado é" );
72      printArray( answer ); /* exibe array ordenado */
73
74      /* exibe elemento mediano */
75      printf( "\n\nA mediana é o elemento %d do\n"
76              "array ordenado de %d elementos.\n"
77              "Para essa execução, a mediana é %d\n\n",
78              SIZE / 2, SIZE, answer[ SIZE / 2 ] );
79  } /* fim da função median */
80
81  /* determina a resposta mais frequente */
82  void mode( int freq[], const int answer[] )
83  {
84      int rating; /* contador para acessar os elementos 1-9 do array freq */
85      int j; /* contador para resumir os elementos 0-98 do array answer */
86      int h; /* contador para exibir histogramas dos elementos no array freq */
87      int largest = 0; /* representa maior frequência */
88      int modeValue = 0; /* representa resposta mais frequente */
89
90      printf( "\n%s\n%s\n%s\n",
91              "*********", "  Moda", "*********" );
92
93      /* inicializa frequências em 0 */
94      for ( rating = 1; rating <= 9; rating++ ) {
95         freq[ rating ] = 0;
96      } /* fim do for */
97
98      /* frequências de resumo */
99      for ( j = 0; j < SIZE; j++ ) {
100        ++freq[ answer[ j ] ];
101     } /* fim do for */
102
103     /* cabeçalhos de impressão para colunas do resultado */
104     printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
105             "Resposta", "Frequência", "Histograma",
106             "1    1    2    2", "5    0    5    0    5" );
107
```

Ordenando Arrays

```
108      /* exibe resultados */
109      for ( rating = 1; rating <= 9; rating++ ) {
110         printf( "%8d%11d ", rating, freq[ rating ] );
111
112         /* acompanha valor da moda e valor da maior frequência */
113         if ( freq[ rating ] > largest ) {
114            largest = freq[ rating ];
115            modeValue = rating;
116         } /* fim do if */
117
118         /* barra de histograma de saída de impressão que representa valor de frequência */
119         for ( h = 1; h <= freq[ rating ]; h++ ) {
120            printf( "*" );
121         } /* fim do for interno */
122
123         printf( "\n" ); /* sendo nova linha de saída */
124      } /* fim do for externo */
125
126      /* exibe o valor da moda */
127      printf( "A moda é o valor mais frequente.\n"
128              "Para essa execução, a moda é %d, que ocorreu"
129              " %d vezes.\n", modeValue, largest );
130   } /* fim da função mode */
131
132   /* função que ordena um array com o algoritmo bubble sort */
```

Figura 6.16 ■ Programa de análise de dados para pesquisa. (Parte 2 de 3.)

Ordenando Arrays

```
133  void bubbleSort( int a[] )
134  {
135     int pass; /* contador de passada */
136     int j; /* contador de comparação */
137     int hold; /* local temporário usado para troca de elementos */
138
139     /* loop para controlar número de passadas */
140     for ( pass = 1; pass < SIZE; pass++ ) {
141
142        /* loop para controlar número de comparações por passada */
143        for ( j = 0; j < SIZE - 1; j++ ) {
144
145           /* troca elementos se estiverem fora de ordem */
146           if ( a[ j ] > a[ j + 1 ] ) {
147              hold = a[ j ];
148              a[ j ] = a[ j + 1 ];
149              a[ j + 1 ] = hold;
150           } /* fim do if */
151        } /* fim do for interno */
152     } /* fim do for externo */
153  } /* fim da função bubbleSort */
154
155  /* imprime conteúdo do array de resultados (20 valores por linha) */
156  void printArray( const int a[] )
157  {
158     int j; /* contador */
159
160     /* imprime conteúdo do array */
161     for ( j = 0; j < SIZE; j++ ) {
162
163        if ( j % 20 == 0 ) { /* inicia nova linha a cada 20 valores */
164           printf( "\n" );
165        } /* fim do if */
166
167        printf( "%2d", a[ j ] );
168     } /* fim do for */
169  } /* fim da função printArray */
```

Figura 6.16 ■ Programa de análise de dados para pesquisa. (Parte 3 de 3.)

Ordenando Arrays

```
*********
Média
*********
A média é o valor médio dos itens de dados.
A média é igual ao total de todos
os itens de dados dividido pelo número
de itens de dados ( 99 ). O valor médio para essa
execução é: 681 / 99 = 6.8788

*********
Mediana
*********

O array não ordenado de respostas é
6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7
```

Figura 6.17 ■ Exemplo de execução para o programa de análise de dados para pesquisa. (Parte 1 de 2.)

Ordenando Arrays

```
O array ordenado é
1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

A mediana é o elemento 49 do
array ordenado de 99 elementos.
Para essa execução, a mediana é 7

*********
Moda
*********
Resposta   Frequência     Histograma
                                  1    1    2    2
                             5    0    5    0    5
        1             1    *
        2             3    ***
        3             4    ****
        4             5    *****
        5             8    ********
        6             9    *********
        7            23    ***********************
        8            27    ***************************
        9            19    *******************
A moda é o valor mais frequente.
Para essa execução, a moda é 8, o que ocorreu 27 vezes.
```

Figura 6.17 ■ Exemplo de execução para o programa de análise de dados para pesquisa. (Parte 2 de 2.)

Estudo de caso: calculando média, mediana e moda usando arrays

- Os computadores normalmente são usados para análise de dados de pesquisa, para compilar e analisar os resultados de pesquisas comuns e pesquisas de opinião.
- A Figura 6.16 usa o array response inicializado com 99 respostas de uma pesquisa.
- Cada resposta corresponde a um número de 1 a 9.
- O programa calcula a média, a mediana e a moda dos 99 valores.
- A média é a média aritmética dos 99 valores.
- A função mean (linha 40) calcula a média totalizando os 99 elementos e dividindo o resultado por 99.
- A mediana é o 'valor intermediário'.

Estudo de caso: calculando média, mediana e moda usando arrays

- A função median (linha 61) determina a mediana chamando a função bubbleSort (definida na linha 133) para classificar o array de respostas em ordem crescente, depois escolhendo o elemento do meio, answer[SIZE / 2], do array classificado.
- Quando existe um número par de elementos, a mediana deve ser calculada como a média dos dois elementos do meio.
- A função median atualmente não oferece essa capacidade.
- A função printArray (linha 156) é chamada para exibir o array response.
- A moda é o valor que ocorre com mais frequência entre as 99 respostas..

Estudo de caso: calculando média, mediana e moda usando arrays

- A função mode (linha 82) determina a moda contando o número de respostas de cada tipo, depois selecionando o valor que aparece mais vezes.
- Essa versão da função mode donão suporta um empate (ver Exercício 6.14).
- A função mode também produz um histograma que ajuda a determinar a moda graficamente.
- A Figura 6.17 contém um exemplo da execução desse programa.

Pesquisando Arrays

- Talvez seja necessário determinar se um array contém um valor que combina com certo valor de chave.
- O processo de encontrar determinado elemento de um array é chamado pesquisa.
- Nesta seção, discutiremos duas técnicas de pesquisa — a pesquisa linear simples e, a mais eficiente (porém mais complexa), pesquisa binária.

Pesquisando Arrays

```
1  /* Figura 6.18: fig06_18.c
2     Pesquisa linear de um array */
3  #include <stdio.h>
4  #define SIZE 100
5
6  /* protótipo de função */
7  int linearSearch( const int array[], int key, int size );
8
9  /* função main inicia a execução do programa */
10 int main( void )
11 {
12    int a[ SIZE ]; /* cria array a */
13    int x; /* contador para inicializar elementos 0-99 do array a */
14    int searchKey; /* valor para localizar no array a */
```

Figura 6.18 ■ Pesquisa linear de um array. (Parte 1 de 2.)

Pesquisando Arrays

```
15      int element; /* variável para manter local de searchKey or -1 */
16
17      /* criar dados */
18      for ( x = 0; x < SIZE; x++ ) {
19         a[ x ] = 2 * x;
20      } /* fim do for */
21
22      printf( "Digite chave de pesquisa de inteiro:\n" );
23      scanf( "%d", &searchKey );
24
25      /* tenta localizar searchKey no array a */
26      element = linearSearch( a, searchKey, SIZE );
27
28      /* exibe resultados */
29      if ( element != -1 ) {
30         printf( "Valor encontrado no elemento %d\n", element );
31      } /* fim do if */
32      else {
33         printf( "Valor não encontrado\n" );
34      } /* fim do else */
35
36      return 0; /* indica conclusão bem-sucedida */
37   } /* fim do main */
38
```

Pesquisando Arrays

```
39  /* Compara chave com cada elemento do array até o local ser encontrado
40     ou até o final do array ser alcançado; retorna subscrito do elemento
41     se chave foi encontrada ou -1 se chave não encontrada */
42  int linearSearch( const int array[], int key, int size )
43  {
44     int n; /* contador */
45
46     /* loop pelo array */
47     for ( n = 0; n < size; ++n ) {
48
49        if ( array[ n ] == key ) {
50           return n; /* retorna local da chave */
51        } /* fim do if */
52     } /* fim do for */
53
54     return -1; /* chave não encontrada */
55  } /* fim da função linearSearch */
```

```
Digite chave de pesquisa de inteiro:
36
Valor encontrado no elemento 18
```

```
Digite chave de pesquisa de inteiro:
37
Valor não encontrado
```

Figura 6.18 ■ Pesquisa linear de um array. (Parte 2 de 2.)

Pesquisando Arrays

- A pesquisa linear (Figura 6.18) compara cada elemento do array com a chave de pesquisa.
- Como o array não está em uma ordem em particular, o valor pode ser encontrado tanto no primeiro elemento quanto no último.
- Na média, portanto, o programa terá de comparar a chave de pesquisa com metade dos elementos do array.

Pesquisando Arrays

- O método de pesquisa linear funciona bem para arrays pequenos ou não ordenados.
- Entretanto, para arrays grandes, a pesquisa linear é ineficaz.
- Se o array estiver ordenado, a técnica de pesquisa binária de alta velocidade poderá ser utilizada.
- O algoritmo de pesquisa binária desconsidera metade dos elementos em um array ordenado após cada comparação.
- O algoritmo localiza o elemento do meio do array e o compara com a chave de pesquisa.

Pesquisando Arrays

- **Se eles são iguais, a chave de pesquisa é encontrada, e o subscrito do array desse elemento é retornado.**
- **Se eles não são iguais, o problema fica reduzido à pesquisa da metade do array.**
- **Se a chave de pesquisa for menor que o elemento do meio do array, a primeira metade do array é pesquisada; caso contrário, a segunda metade do array é pesquisada.**
- **Se a chave de pesquisa não é encontrada no subarray especificado (parte do array original), o algoritmo é repetido em um quarto do array original.**

Pesquisando Arrays

- A pesquisa continua até que a chave de pesquisa seja igual ao elemento do meio do subarray, ou até que o subarray consista em um elemento que não seja igual à chave de pesquisa (ou seja, a chave de pesquisa não foi localizada).
- No cenário da pior das hipóteses, a pesquisa de um array de 1023 elementos exige apenas 10 comparações usando uma pesquisa binária.
- Dividir 1024 repetidamente por 2 gera os valores 512, 256, 128, 64, 32, 16, 8, 4, 2 e 1.
- O número 1024 (2^{10}) é dividido por 2 apenas 10 vezes para chegar ao valor 1.
- Dividir por 2 é equivalente a uma comparação no algoritmo de pesquisa binária.

Pesquisando Arrays

- Um array de 1048576 (2^{20}) elementos exige um máximo de 20 comparações para que a chave de pesquisa seja encontrada.
- Um array de um bilhão de elementos exige no máximo 30 comparações para que a chave de pesquisa seja encontrada.
- É um progresso tremendo em termos de desempenho em relação à pesquisa linear, que exige a comparação de uma chave de pesquisa a uma média de metade dos elementos do array.
- Para um array de um bilhão de elementos, esta é uma diferença entre uma média de 500 milhões de comparações e um máximo de 30 comparações!

Pesquisando Arrays

```c
/* Figura 6.19: fig06_19.c
   Pesquisa binária de um array ordenado */
#include <stdio.h>
#define SIZE 15

/* protótipo de funções */
int binarySearch( const int b[], int searchKey, int low, int high );
void printHeader( void );
void printRow( const int b[], int low, int mid, int high );

/* função main inicia a execução do programa */
int main( void )
   {
   int a[ SIZE ]; /* cria array a */
   int i; /* contador para inicializar elementos 0-14 do array a */
   int key; /* valor a localizar no array a */
   int result; /* variável para manter local da chave ou -1 */

   /* cria dados */
   for ( i = 0; i < SIZE; i++ ) {
      a[ i ] = 2 * i;
   } /* fim do for */

   printf( "Digite um número entre 0 e 28: " );
   scanf( "%d", &key );

   printHeader();

   /* procura chave no array a */
   result = binarySearch( a, key, 0, SIZE - 1 );

   /* mostra resultados */
   if ( result != -1 ) {
```

Figura 6.19 ■ Pesquisa binária de um array ordenado. (Parte 1 de 3.)

Pesquisando Arrays

```
34          printf( "\n%d encontrados no elemento de array %d\n", key, result );
35       } /* fim do if */
36       else {
37          printf( "\n%d não encontrados\n", key );
38       } /* end else */
39
40       return 0; /* indica conclusão bem-sucedida */
41    } /* fim do main */
42
43    /* função para realizar pesquisa binária de um array */
44    int binarySearch( const int b[], int searchKey, int low, int high )
45    {
46       int middle; /* variável para manter elemento do meio do array */
47
48       /* loop até subscrito baixo ser maior que o subscrito alto */
49       while ( low <= high ) {
50
51          /* determina elemento do meio do subarray sendo pesquisado */
52          middle = ( low + high ) / 2;
53
54          /* exibe subarray usado nessa iteração de loop */
55          printRow( b, low, middle, high );
56
57          /* se searchKey combinou com elemento do meio, retorna middle */
58          if ( searchKey == b[ middle ] ) {
59             return middle;
60          } /* fim do if */
61
62          /* se searchKey menor que o elemento do meio, define novo high */
63          else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* procura extremidade baixa do array */
65          } /* fim do else if */
66
67          /* se searchKey maior que o elemento do meio, define novo low */
68          else {
69             low = middle + 1; /* procura extremidade alta do array */
70          } /* fim do else */
71       } /* fim do while */
72
```

Pesquisando Arrays

```
73      return -1; /* searchKey não encontrada */
74   } /* fim da função binarySearch */
75
76   /* Imprime cabeçalho para a saída */
77   void printHeader( void )
78   {
79      int i; /* contador */
80
81      printf( "\nSubscritos:\n" );
82
83      /* cabeçalho da coluna de saída */
84      for ( i = 0; i < SIZE; i++ ) {
85         printf( "%3d ", i );
86      } /* fim do for */
87
88      printf( "\n" ); /* inicia nova linha de saída */
89
90      /* linha de saída de caracteres */
91      for ( i = 1; i <= 4 * SIZE; i++ ) {
92         printf( "-" );
93      } /* fim do for */
94
95      printf( "\n" ); /* inicia nova linha de saída */
96   } /* fim da função printHeader */
97
98   /* Imprime uma linha de saída mostrando a parte atual
```

Figura 6.19 ■ Pesquisa binária de um array ordenado. (Parte 2 de 3.)

Pesquisando Arrays

```
 99     do array sendo processado. */
100  void printRow( const int b[], int low, int mid, int high )
101  {
102     int i; /* contador para percorrer o array b */
103
104     /* loop pelo array inteiro */
105     for ( i = 0; i < SIZE; i++ ) {
106
107        /* mostra espaços se for a da faixa atual do subarray */
108        if ( i < low || i > high ) {
109           printf( "   " );
110        } /* fim do if */
111        else if ( i == mid ) { /* mostra elemento do meio */
112           printf( "%3d*", b[ i ] ); /* marca valor do meio */
113        } /* fim do else if */
114        else { /* mostra outros elementos no subarray */
115           printf( "%3d ", b[ i ] );
116        } /* fim do else */
117     } /* fim do for */
118
119     printf( "\n" ); /* inicia nova linha de saída */
120  } /* fim da função printRow */
```

Pesquisando Arrays

```
Digite um número entre 0 e 28: 25
Subscritos:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                              16  18  20  22* 24  26  28
                                          24  26* 28
                                          24*
25 não encontrado
```

```
Digite um número entre 0 e 28: 8
Subscritos:
  0   1   2   3   4   5   6   7   8   9   10   11   12   13   14
-------------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18   20   22   24   26   28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*
8 encontrado no elemento de array 4
```

```
Digite um número entre 0 e 28: 6
Subscritos:
  0   1   2   3   4   5   6   7   8   9   10   11   12   13   14
-------------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18   20   22   24   26   28
  0   2   4   6*  8  10  12
6 encontrado no elemento de array 3
```

Figura 6.19 ■ Pesquisa binária de um array ordenado. (Parte 3 de 3.)

Pesquisando Arrays

- As comparações máximas para qualquer array podem ser determinadas ao se encontrar a primeira potência de 2 maior que o número de elementos do array.
- A Figura 6.19 apresenta a versão iterativa da função binarySearch (linhas 44–74).
- A função recebe quatro argumentos — um array inteiro b a ser pesquisado, um inteiro searchKey, o subscrito de array low e o subscrito de array high (eles definem a parte do array a ser pesquisada).
- Se a chave de pesquisa não combina com o elemento do meio de um subarray, o subscrito low ou o subscrito high é modificado, de modo que um subarray menor possa ser pesquisado.

Pesquisando Arrays

- Se a chave de pesquisa for menor que o elemento do meio, o subscrito high é definido como middle - 1, e a pesquisa continua nos elementos delow até middle - 1.
- Se a chave de pesquisa for maior que o elemento do meio, o subscritolow é definido como middle + 1, e a pesquisa continua nos elementos de middle + 1 até high.
- O programa usa um array de 15 elementos.
- A primeira potência de 2 maior que o número de elementos nesse array é 16 (24), de modo que são necessárias, no máximo, 4 comparações para que a chave de pesquisa seja encontrada.

Pesquisando Arrays

- O programa usa a função printHeader (linhas 77-96) para mostrar os subscritos do array, e a função printRow (linhas 100-120) para mostrar cada subarray durante o processo de pesquisa binária.
- O elemento do meio em cada subarray é marcado com um asterisco (*) para indicar o elemento ao qual a chave de pesquisa será comparada.

Arrays multidimensionais

- Os arrays em C podem ter vários subscritos.
- Os arrays de subscritos múltiplos (também chamados arrays multidimensionais) podem ser usados na representação de tabelas de valores que consistem em informações organizadas em linhas e colunas.
- Para identificar determinado elemento de tabela, devemos especificar dois subscritos: o primeiro (por convenção) identifica a linha do elemento, e o segundo (por convenção) identifica a coluna do elemento.
- Tabelas ou arrays que exigem dois subscritos para identificar determinado elemento são chamados arrays de subscrito duplo.

Arrays multidimensionais

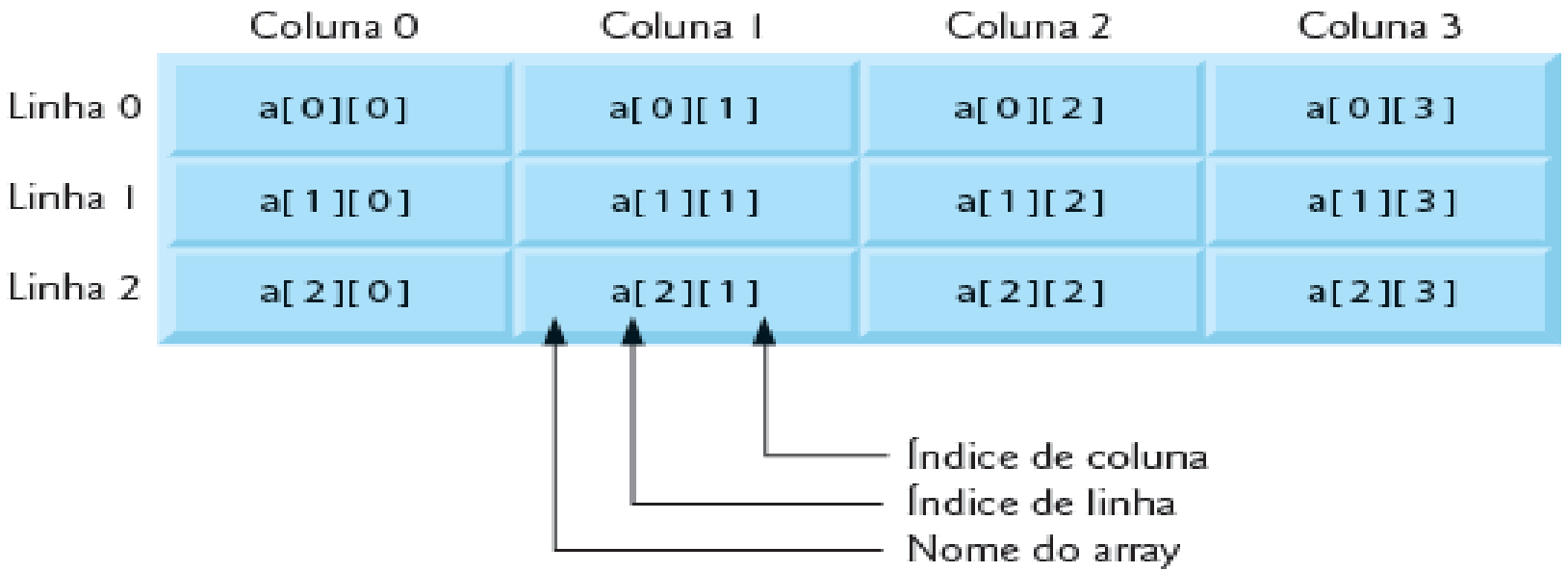


Figura 6.20 ■ Array de subscrito duplo com três linhas e quatro colunas.

Arrays multidimensionais

- Arrays de subscritos múltiplos podem ter mais de dois subscritos.
- A Figura 6.20 ilustra um array de subscrito duplo, a.
- O array contém três linhas e quatro colunas, de modo que é considerado um array de 3 por 4.
- Em geral, um array com *m* linhas e *n* colunas é chamado array *m*-por-*n*.

Arrays multidimensionais

- Cada elemento no array a é identificado na Figura 6.20 por um nome de elemento na forma a[i][j]; a é o nome do array, e i e j são os subscritos que identificam de forma única cada elemento em a.
- Todos os nomes dos elementos na primeira linha têm um primeiro subscrito 0; todos os nomes dos elementos na quarta coluna têm um segundo subscrito 3.

Arrays multidimensionais

Erro comum de programação 6.9

Referenciar um elemento de array de subscrito duplo como a[x, y] *em vez de* a[x][y]*. C interpreta* a[x, y] *como* a[y]*, e, dessa forma, isso não causa um erro de compilação.*

Arrays multidimensionais

- Um array de subscritos múltiplos pode ser inicializado quando definido, assim como um array de único subscrito.
- Por exemplo, um array de subscrito duplo int b[2][2] poderia ser definido e inicializado com
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
- Os valores são agrupados por linha em chaves.
- Os valores no primeiro conjunto de chaves inicializam a linha 0, e os valores no segundo conjunto de chaves inicializam a linha 1.
- Assim, os valores 1 e 2 inicializam os elementos b[0][0] e b[0][1], respectivamente, e os valores 3 e 4 inicializam os elementos b[1][0] e b[1][1], respectivamente.

Arrays multidimensionais

- ***Se não houver inicializadores suficientes para determinada linha, os elementos restantes dessa linha serão inicializados em 0.***
- **Assim,**
  - `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`

inicializaria b[0][0] em 1, b[0][1] em 0, b[1][0] em 3 e b[1][1] em 4.

Arrays multidimensionais

```
1  /* Figura 6.21: fig06_21.c
2     Inicializando arrays multidimensionais */
3  #include <stdio.h>
4
5  void printArray( const int a[][ 3 ] ); /* protótipo de função */
6
7  /* função main inicia a execução do programa */
8  int main( void )
9  {
10    /* inicializa array1, array2, array3 */
11    int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12    int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
13    int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14
15    printf( "Valores em array1 por linha são:\n" );
16    printArray( array1 );
17
18    printf( "Valores em array2 por linha são:\n" );
19    printArray( array2 );
20
21    printf( "Valores em array3 por linha são:\n" );
22    printArray( array3 );
23    return 0; /* indica conclusão bem-sucedida */
24 } /* fim do main */
25
```

Arrays multidimensionais

```
26  /* função para mostrar array com duas linhas e três colunas */
27  void printArray( const int a[][ 3 ] )
28  {
29     int i; /* contador de linha */
30     int j; /* contador de coluna */
31
32     /* loop pelas linhas */
33     for ( i = 0; i <= 1; i++ ) {
34
35        /* imprime valores nas colunas */
36        for ( j = 0; j <= 2; j++ ) {
37           printf( "%d ", a[ i ][ j ] );
38           } /* fim do for interno */
39
40        printf( "\n" ); /* inicia nova linha de resultados */
41     } /* fim do for externo */
42  } /* fim da função printArray */
```

```
Os valores em array1 por linha são:
1 2 3
4 5 6
Os valores em array2 por linha são:
1 2 3
4 5 0
Os valores em array3 por linha são:
1 2 0
4 0 0
```

Figura 6.21 ■ Inicializando arrays multidimensionais.

Arrays multidimensionais

- A Figura 6.21 demonstra a definição e a inicialização de arrays de subscrito duplo.
- O programa define três arrays de duas linhas e três colunas (seis elementos cada).
- A definição de array1 (linha 11) oferece seis inicializadores em duas sublistas.
- A primeira sublista inicializa a primeira linha (ou seja, a linha 0) do array com os valores 1, 2 e 3; e a segunda sublista inicializa a segunda linha (ou seja, a linha 1) do array com os valores 4, 5 e 6.

Arrays multidimensionais

- Se as chaves ao redor de cada sublista forem removidas da lista de inicializadores de array1 o compilador inicializará os elementos da primeira linha seguidos pelos elementos da segunda linha.
- A definição dearray2 (linha 12) oferece cinco inicializadores.
- Os inicializadores são atribuídos à primeira linha, e depois à segunda linha.
- Quaisquer elementos que não possuam um inicializador explícito são automaticamente inicializados em zero, de modo que array2[1][2] é inicializado em 0.
- A definição de array3 (linha 13) oferece três inicializadores em duas sublistas.

Arrays multidimensionais

- A sublista para a primeira linha inicializa explicitamente os dois primeiros elementos da primeira linha em 1 e 2.
- O terceiro elemento é inicializado em zero.
- A sublista para a segunda linha inicializa explicitamente o primeiro elemento em 4.
- Os dois últimos elementos são inicializados em zero.
- O programa chama printArray (linhas 27-43) para mostrar os elementos de cada array.
- A definição de função especifica o parâmetro de array como const int a[][3].
- Quando recebemos um array de subscrito único como parâmetro, os colchetes do array estão vazios na lista de parâmetros da função.

Arrays multidimensionais

- O primeiro subscrito de um array de subscritos múltiplos também não é exigido, mas todos os subscritos seguintes, sim.
- O compilador usa esses subscritos para determinar os locais na memória dos elementos nos arrays de subscritos múltiplos.
- Todos os elementos do array são armazenados consecutivamente na memória, independentemente do número de subscritos.
- Em um array de subscrito duplo, a primeira linha é armazenada na memória seguida pela segunda linha.
- Oferecer os valores de subscrito em uma declaração de parâmetros permite que o compilador informe à função como localizar um elemento no array.

Arrays multidimensionais

- Em um array de subscrito duplo, cada linha é, basicamente, um array de subscrito único.
- Para localizar um elemento em determinada linha, o compilador deve saber quantos elementos há em cada linha, para que possa pular o número apropriado de locais da memória ao acessar o array.
- Assim, ao acessar a[1][2] em nosso exemplo, o compilador sabe que deve 'pular' os três elementos da primeira linha para chegar à segunda linha (linha 1).
- Depois, o compilador acessa o terceiro elemento dessa linha (elemento 2).

Arrays multidimensionais

- Muitas manipulações comuns de array usam as estruturas de repetição for.
- Por exemplo, a estrutura a seguir define todos os
- elementos na terceira linha do array a da Figura 6.20 como zero:
  - ```
    for ( coluna = 0; coluna <= 3; coluna++ ) {
       a[ 2 ][ coluna ] = 0;
    }
    ```
- Especificamos a terceira linha, e, portanto, sabemos que o primeiro subscrito é sempre 2 (novamente, 0 é a primeira linha e 1 é a segunda).
    ```

Arrays multidimensionais

- O loop varia apenas o segundo subscrito (ou seja, a coluna).
- A estrutura for anterior é equivalente aos comandos de atribuição:
  - a[2][0] = 0;
    a[2][1] = 0;
    a[2][2] = 0;
    a[2][3] = 0;

Arrays multidimensionais

- A seguinte estrutura for aninhada determina o total de todos os elementos no array a.
  - total = 0;
  - ```
    for ( linha = 0; linha <= 2; linha++ ) {
      for ( coluna = 0; coluna <= 3; coluna++ ) {
        total += a[ linha ][ coluna ];
      }
    }
    ```
- O comando for totaliza os elementos do array uma linha de cada vez.
    ```

Arrays multidimensionais

- A estrutura for externa começa na definição de row (ou seja, o subscrito da linha) em 0, de modo que os elementos da primeira linha possam ser totalizados pela estrutura for.
- A estrutura for externa, então, incrementa row em 1, de modo que os elementos da segunda linha possam ser totalizados.
- Então, a estrutura for externa incrementa row em 2, para que os elementos da terceira linha possam ser totalizados.
- O resultado é impresso quando a estrutura for aninhada termina.

Arrays multidimensionais

```
1  /* Figura 6.22: fig06_22.c
2     Exemplo de array com subscrito duplo */
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  /* protótipo de funções */
8  int minimum( const int grades[][ EXAMS ], int pupils, int tests );
9  int maximum( const int grades[][ EXAMS ], int pupils, int tests );
10 double average( const int setOfGrades[], int tests );
11 void printArray( const int grades[][ EXAMS ], int pupils, int tests );
12
13 /* função main inicia a execução do programa */
14 int main( void )
15    {
16    int student; /* contador de alunos */
17
```

Figura 6.22 ■ Exemplo de arrays com subscrito duplo. (Parte 1 de 3.)

Arrays multidimensionais

```
18     /* inicializa notas de aluno para três alunos (linhas) */
19     const int studentGrades[ STUDENTS ][ EXAMS ] =
20        { { 77, 68, 86, 73 },
21          { 96, 87, 89, 78 },
22          { 70, 90, 86, 81 } };
23
24     /* mostra array studentGrades */
25     printf( "O array é:\n" );
26     printArray( studentGrades, STUDENTS, EXAMS );
27
28     /* determina menor e maior valor de nota */
29     printf( "\n\nMenor nota: %d\nMaior nota: %d\n",
30        minimum( studentGrades, STUDENTS, EXAMS ),
31        maximum( studentGrades, STUDENTS, EXAMS ) );
32
33     /* calcula nota média de cada aluno */
34     for ( student = 0; student < STUDENTS; student++ ) {
35        printf( "A nota média do aluno %d é %.2f\n",
36           student, average( studentGrades[ student ], EXAMS ) );
37     } /* fim do for */
38
39     return 0; /* indica conclusão bem-sucedida */
40  } /* fim do main */
41
42  /* Encontra a menor nota */
43  int minimum( const int grades[][ EXAMS ], int pupils, int tests )
44  {
45     int i; /* contador de alunos */
46     int j; /* contador de exames */
47     int lowGrade = 100; /* inicializa para maior nota possível */
48
49     /* loop pelas linhas de notas */
50     for ( i = 0; i < pupils; i++ ) {
51
```

Arrays multidimensionais

```
52          /* loop pelas colunas de notas */
53          for ( j = 0; j < tests; j++ ) {
54
55             if ( grades[ i ][ j ] < lowGrade ) {
56                lowGrade = grades[ i ][ j ];
57             } /* fim do if */
58          } /* fim do for interno */
59       } /* fim do for externo */
60
61       return lowGrade; /* retorna menor nota */
62    } /* fim da função minimum */
63
64    /* Acha a maior nota */
65    int maximum( const int grades[][ EXAMS ], int pupils, int tests )
66    {
67       int i; /* contador de alunos */
68       int j; /* contador de exames */
69       int highGrade = 0; /* inicializa para menor nota possível */
70
71       /* loop pelas linhas de notas */
72       for ( i = 0; i < pupils; i++ ) {
73
74          /* loop pelas colunas de notas */
75          for ( j = 0; j < tests; j++ ) {
```

Figura 6.22 ■ Exemplo de arrays com subscrito duplo. (Parte 2 de 3.)

Arrays multidimensionais

```
76
77             if ( grades[ i ][ j ] > highGrade ) {
78                highGrade = grades[ i ][ j ];
79             } /* fim do if */
80          } /* fim do for interno */
81       } /* fim do for externo */
82
83       return highGrade; /* retorna nota máxima */
84    } /* fim da função maximum */
85
86    /* Determina a nota média para determinado aluno */
87    double average( const int setOfGrades[], int tests )
88    {
89       int i; /* contador de exame */
90       int total = 0; /* soma das notas de teste */
91
92       /* soma todas as notas para um aluno */
93       for ( i = 0; i < tests; i++ ) {
94          total += setOfGrades[ i ];
95       } /* fim do for */
96
97       return ( double ) total / tests; /* média */
98    } /* fim da função average */
99
100   /* Mostra o array */
101   void printArray( const int grades[][ EXAMS ], int pupils, int tests )
102   {
103      int i; /* contador de aluno */
104      int j; /* contador de exame */
105
106      /* mostra cabeçalhos de coluna */
107      printf( "                            [0] [1] [2] [3]" );
108
109      /* mostra notas em formato tabular */
110      for ( i = 0; i < pupils; i++ ) {
111
```

Arrays multidimensionais

```
112         /* mostra label para linha */
113         printf( "\nstudentGrades[%d] ", i );
114
115         /* mostra notas para um aluno */
116         for ( j = 0; j < tests; j++ ) {
117            printf( "%-5d", grades[ i ][ j ] );
118         } /* fim do for interno */
119      } /* fim do for externo */
120   } /* fim da função printArray */
```

```
O array é:
                  [0] [1] [2] [3]
studentGrades[0] 77  68  86  73
studentGrades[1] 96  87  89  78
studentGrades[2] 70  90  86  81

Menor nota: 68
Maior nota: 96
A nota média do aluno 0 é 76,00
A nota média do aluno 1 é 87,50
A nota média do aluno 2 é 81,75
```

Figura 6.22 ■ Exemplo de arrays com subscrito duplo. (Parte 3 de 3.)

Arrays multidimensionais

- A Figura 6.22 realiza várias outras manipulações comuns de array sobre o array 3 por 4 studentGrades usando estruturas for.
- Cada linha do array representa um aluno, e cada coluna representa uma nota em um dos quatro exames que os alunos realizaram durante o semestre.
- As manipulações de array são realizadas por quatro funções.
- A função minimum (linhas 43-62) determina a nota mais baixa de qualquer aluno durante o semestre.

Arrays multidimensionais

- A função maximum (linhas 65-84) determina a nota mais alta de qualquer aluno durante o semestre.
- A função average (linhas 87-98) determina a média do semestre de um aluno em particular.
- A função printArray (linhas 101-120) mostra o array com subscrito duplo em um formato tabular, bem-arrumado.

Arrays multidimensionais

- As funções minimum, maximum e printArray recebem três argumentos cada uma — o array studentGrades (chamado grades em cada função), o número de alunos (linhas do array) e o número de exames (colunas do array).
- Cada função percorre o array grades usando estruturas for aninhadas.

Arrays multidimensionais

- A estrutura for aninhada a seguir vem da definição da função minimum :

```
/* loop pelas linhas de notas */
for ( i = 0; i < pupils; i++ ) {
   /* loop pelas colunas de notas */
   for ( j = 0; j < tests; j++ ) {
      if ( grades[ i ][ j ] < lowGrade ) {
         lowGrade = grades[ i ][ j ];
      } /* fim do if */
   } /* fim do for interno */
} /* fim do for externo */
```

Arrays multidimensionais

- A estrutura for externa começa com a definição de i (ou seja, o subscrito de linha) como 0, de modo que os elementos da primeira linha (ou seja, as notas do primeiro aluno) possam ser comparados com a variável lowGrade no corpo da estrutura for interna.
- A estrutura for interna percorre as quatro notas de determinada linha e compara cada nota com lowGrade.
- Se uma nota for menor que lowGrade, lowGrade é definido como essa nota.
- A estrutura for externa, então, incrementa o subscrito da linha em 1.
- Os elementos da segunda linha são comparados com a variável lowGrade.

Arrays multidimensionais

- A estrutura for externa, então, incrementa o subscrito de linha em 2.
- Os elementos da terceira linha são comparados com a variável lowGrade.
- Quando a execução da estrutura aninhada termina, lowGrade contém a menor nota no array de subscrito duplo.
- A função maximum funciona de modo semelhante à função minimum.
- A função average (linha 87) usa dois argumentos — um array de subscrito único de resultados de teste para determinado aluno, chamado setOfGrades, e o número de resultados dos testes no array.

Arrays multidimensionais

- Quando average é chamada, o primeiro argumento studentGrades[student] é passado.
- Isso faz com que o endereço de uma linha do array de subscrito duplo seja passado para average.
- O argumento studentGrades[1] é o endereço inicial da segunda linha do array.
- Lembre-se de que o array com subscrito duplo é, basicamente, um array de arrays de subscrito único, e que o nome do array de subscrito único é o endereço do array na memória.
- A função average calcula a soma dos elementos do array, divide o total pelo número de resultados de teste e retorna o resultado de ponto flutuante.

Lembre-se

“ Código é como humor. Quando você tem que explicar, é ruim. ” Cory House