Técnicas de Programação

TP0701

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 usar as funções da biblioteca de tratamento de caracteres (<ctype.h>).
- A usar as funções de conversão de strings da biblioteca de utilitários gerais (<stdlib.h>).
- A usar as funções de entrada/saída de string e caracteres da biblioteca-padrão de entrada/saída (<stdio.h>).
- A usar as funções de processamento de string da biblioteca de tratamento de strings (<string.h>).
- O poder das bibliotecas de funções que levam à reutilização de software.

Introdução

- Apresentaremos as funções da biblioteca-padrão de C que facilitam o processamento de strings e caracteres.
- As funções permitem que os programas processem caracteres, strings, linhas de texto e blocos de memória.
- Discutiremos as técnicas usadas para desenvolver editores, processadores de textos, software de layout de página, sistemas computadorizados de composição de textos e outros tipos de software de processamento de textos.
- As manipulações de texto realizadas por funções de entrada/saída formatada, como printf e scanf, podem ser implementadas utilizando-se as funções abordadas neste capítulo.

Fundamentos de strings e caracteres

- Os caracteres são os blocos de montagem fundamentais dos programas-fonte.
- Cada programa é composto por uma sequência de caracteres que, ao serem agrupados de modo significativo, são interpretados pelo computador como uma série de instruções a serem usadas na realização de uma tarefa.
- Um programa pode conter constantes de caractere.
- Uma constante de caractere é um valor int representado por um caractere entre aspas simples.
- O valor de uma constante de caractere é o valor inteiro do caractere no conjunto de caracteres da máquina.

Fundamentos de strings e caracteres

- Por exemplo, 'z' representa o valor inteiro de z, e '\n' o valor inteiro do caractere de nova linha (122 e 10 em ASCII, respectivamente).
- Uma string consiste em uma série de caracteres tratados como uma única entidade.
- Uma string pode incluir letras, dígitos e diversos caracteres especiais, como +, -, *, / e $. Strings literais, ou constantes string, em C são escritas entre aspas.

Fundamentos de strings e caracteres

- Uma string em C é um array de caracteres que termina no caractere nulo ('\0').
- Strings são acessadas por meio de um ponteiro para o primeiro caractere da string.
- O valor de uma string é o endereço de seu primeiro caractere.
- Assim, em C, é apropriado dizer que uma string é um ponteiro — na verdade, um ponteiro para o primeiro caractere da string.
- Nesse sentido, as strings são como arrays, pois um array também é um ponteiro para o seu primeiro elemento.
- Um array de caracteres, ou uma variável do tipo char *, pode ser inicializado com uma string em uma definição.

Fundamentos de strings e caracteres

- As definições
  - char color[] = “azul”;
    const char *corPtr = “azul”;

  inicializa uma variável como a string “azul”.
- A primeira definição cria um array de 5 elementos cor que contêm os caracteres'b', 'l', 'u', 'e' e '\0'.
- A segunda definição cria a variável de ponteiro corPtr, que aponta para a string “azul” em algum lugar da memória.

Fundamentos de strings e caracteres

Dica de portabilidade 8.1

Quando uma variável do tipo char * *é inicializada com uma string literal, alguns compiladores podem colocar a string em um local da memória onde ela não possa ser modificada. Se você acha que terá de modificar uma string literal, ela deverá ser armazenada em um array de caracteres para garantir que a modificação seja possível em todos os sistemas.*

Fundamentos de strings e caracteres

- A definição anterior de array também poderia ter sido escrita como
  - `char cor[] = { 'a', 'z', 'u', 'l', '\0' };`
- Ao definir um array de caracteres para que contenha uma string, ele precisa ser grande o suficiente para armazenar a string e seu caractere nulo de finalização.
- A definição anterior automaticamente determina o tamanho do array com base no número de inicializadores na lista de inicializadores.

Fundamentos de strings e caracteres

Erro comum de programação 8.1
Não alocar espaço suficiente em um array de caracteres para que ele armazene o caractere nulo que indica o fim de uma string é um erro.

Erro comum de programação 8.2
Imprimir uma 'string' que não contenha um caractere nulo de finalização é um erro.

Fundamentos de strings e caracteres

Dica de prevenção de erro 8.1

Ao armazenar uma string de caracteres em um array de caracteres, cuide para que o array seja grande o suficiente para manter a maior string a ser armazenada. C permite que strings de qualquer tamanho sejam armazenadas. Se uma string for maior que o array de caracteres em que ela deve ser armazenada, os caracteres que ultrapassarem o final do array sobrescreverão dados em posições da memória que sucederem o array.

Fundamentos de strings e caracteres

- Uma string pode ser armazenada em um array utilizando-se scanf.
- Por exemplo, a instrução a seguir armazena uma string no array de caracteres word[20]:
  - scanf("%s", word);
- A string digitada pelo usuário é armazenada em word.
- A variável word é um array, o qual, naturalmente, é um ponteiro, de modo que não é necessário usar o & com o argumento word.
- scanf lerá caracteres até que se encontre espaço, tabulação, nova linha ou indicador de fim de arquivo.
- Assim, é possível que a entrada do usuário exceda 19 caracteres,e que seu programa falhe!

Fundamentos de strings e caracteres

- Por esse motivo, use o especificador de conversão %19s, de modo que scanf possa ler até 19 caracteres, e deixe o último caractere para o caractere nulo de finalização.
- Isso impede que scanf escreva caracteres na memória que ultrapassem o s.
- (Para a leitura de linhas de entrada de qualquer tamanho, existe a função fora do padrão — embora bastante aceita — readline, normalmente incluída em stdio.h.)
- Para que um array de caracteres seja impresso como uma string, o array precisa conter um caractere nulo de finalização.

Fundamentos de strings e caracteres

Erro comum de programação 8.3
Processar um único caractere como se fosse uma string. Uma string é um ponteiro — provavelmente, um inteiro
razoavelmente grande. Porém, um caractere é um inteiro pequeno (valores ASCII na faixa de 0 a 255). Isso provoca
erros em muitos sistemas, pois os endereços de memória baixos são reservados para finalidades especiais como os tratadores de interrupção do sistema operacional; isso causa 'violações de acesso'.

Erro comum de programação 8.4
Passar um caractere para uma função como se fosse um argumento, quando uma string é esperada (e vice-versa),
consiste em um erro de compilação.

Biblioteca de tratamento de caracteres

Protótipo	Descrição da função
`int isdigit( int c );`	Retorna um valor diferente de zero que é considerado verdadeiro se c for um dígito, e 0 (falso) caso contrário.
`int isalpha( int c );`	Retorna um valor diferente de zero que é considerado verdadeiro se c for uma letra, e 0 caso contrário.
`int isalnum( int c );`	Retorna um valor diferente de zero que é considerado verdadeiro se c for um dígito ou uma letra, e 0 caso contrário.
`int isxdigit( int c );`	Retorna um valor verdadeiro se c for um caractere de dígito hexadecimal, e 0 caso contrário. (Veja no Apêndice C: Sistemas Numéricos uma explicação detalhada dos números binários, números octais, números decimais e números hexadecimais.)
`int islower( int c );`	Retorna um valor verdadeiro se c for uma letra minúscula, e 0 caso contrário.
`int isupper( int c );`	Retorna um valor verdadeiro se c for uma letra maiúscula, e 0 caso contrário.
`int tolower( int c );`	Se c for uma letra maiúscula, `tolower` retornará c como uma letra minúscula. Caso contrário, `tolower` retornará o argumento inalterado.
`int toupper( int c );`	Se c for uma letra minúscula, `toupper` retornará c como uma letra minúscula. Caso contrário, `toupper` retornará o argumento inalterado.
`int isspace( int c );`	Retorna um valor verdadeiro se c for um caractere de espaço em branco — nova linha (`'\n'`), espaço (`' '`), form feed (`'\f'`), carriage return (`'\r'`), tabulação horizontal (`'\t'`) ou tabulação vertical (`'\v'`) —, e 0 caso contrário.
`int iscntrl( int c );`	Retorna um valor verdadeiro se c for um caractere de controle, e 0 caso contrário.
`int ispunct( int c );`	Retorna um valor verdadeiro se c for um caractere imprimível diferente de um espaço, de um dígito ou de uma letra, e retorna 0 caso contrário.
`int isprint( int c );`	Retorna um valor verdadeiro se c for um caractere imprimível que inclui um espaço (`' '`), e retorna 0 caso contrário.
`int isgraph( int c );`	Retorna um valor verdadeiro se c for um caractere imprimível diferente de um espaço (`' '`), e retorna 0 caso contrário.

Figura 8.1 ■ Funções da biblioteca de tratamento de caracteres (<ctype.h>).

Biblioteca de tratamento de caracteres

- A **biblioteca de tratamento de caracteres** (<ctype.h>) inclui várias funções que realizam testes úteis e manipulações de dados de caractere.
- Cada função recebe um caractere — representado como um int — ou um EOF como argumento.
- EOF geralmente tem o valor –1, e algumas arquiteturas de hardware não permitem que valores negativos sejam armazenados em variáveis char , de modo que as funções de tratamento de caractere manipulam caracteres como inteiros.
- A Figura 8.1 resume as funções da biblioteca de tratamento de caracteres.

Biblioteca de tratamento de caracteres

```
1  /* Fig. 8.2: fig08_02.c
2     Usando funções isdigit, isalpha, isalnum e isxdigit */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main( void )
7  {
8     printf( "%s\n%s%s\n%s%s\n\n", "De acordo com isdigit: ",
9        isdigit( '8' ) ? "8 é um " : "8 não é um ", "dígito",
10       isdigit( '#' ) ? "# é um " : "# não é um ", "dígito" );
11
12    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
13       "De acordo com isalpha:",
14       isalpha( 'A' ) ? "A é uma " : "A não é uma ", "letra",
15       isalpha( 'b' ) ? "b é uma " : "b não é uma ", "letra",
16       isalpha( '&' ) ? "& é uma " : "& não é uma ", "letra",
17       isalpha( '4' ) ? "4 é uma " : "4 não é uma ", "letra" );
18
```

Biblioteca de tratamento de caracteres

```
19      printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
20         "De acordo com isalnum:",
21         isalnum( 'A' ) ? "A é um " : "A não é um ",
22         "dígito ou uma letra",
23         isalnum( '8' ) ? "8 é um " : "8 não é um ",
24         "dígito ou uma letra",
25         isalnum( '#' ) ? "# é um " : "# não é um ",
26         "dígito ou uma letra" );
27
28      printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
29         "De acordo com isxdigit:",
30         isxdigit( 'F' ) ? "F é um " : "F não é um ",
31         "dígito hexadecimal",
32         isxdigit( 'J' ) ? "J é um " : "J não é um ",
33         "dígito hexadecimal",
34         isxdigit( '7' ) ? "7 é um " : "7 não é um ",
35         "dígito hexadecimal",
36         isxdigit( '$' ) ? "$ é um " : "$ não é um ",
37         "dígito hexadecimal",
38         isxdigit( 'f' ) ? "f é um " : "f não é um ",
39         "dígito hexadecimal" );
40      return 0; /* indica conclusão bem-sucedida */
41   } /* fim do main */
```

Figura 8.2 ■ Exemplo do uso de isdigit, isalpha, isalnum e isxdigit. (Parte 1 de 2.)

Biblioteca de tratamento de caracteres

```
De acordo com isdigit:
8 é um dígito
# não é um dígito

De acordo com isalpha:
A é uma letra
b é uma letra
& não é uma letra
4 não é uma letra

De acordo com isalnum:
A é um dígito ou uma letra
8 é um dígito ou uma letra
# não é um dígito e nem uma letra

De acordo com isxdigit:
F é um dígito hexadecimal
J não é um dígito hexadecimal
7 é um dígito hexadecimal
$ não é um dígito hexadecimal
f é um dígito hexadecimal
```

Figura 8.2 ■ Exemplo do uso de `isdigit`, `isalpha`, `isalnum` e `isxdigit`. (Parte 2 de 2.)

Biblioteca de tratamento de caracteres

- A Figura 8.2 demonstra as funções isdigit, isalpha, isalnum e isxdigit.
- A função isdigit determina se seu argumento é um dígito (0–9).
- A função isalpha estabelece se seu argumento é uma letra maiúscula (A–Z) ou minúscula (a–z).
- A função isalnum determina se seu argumento é uma letra maiúscula, uma letra minúscula ou um dígito.
- A função isxdigit determina se seu argumento é um dígito hexadecimal (A–F, a–f, 0–9).

Biblioteca de tratamento de caracteres

- A Figura 8.2 usa o operador condicional (?:) para determina se a string “ é um ” ou se a string “ não é um ” devem ser impressas na saída de cada caractere testado.
- Por exemplo, a expressão
  - isdigit('8') ? "8 é um " : "8 não é um "

  indica que se '8' for um dígito, a string "8 é um " será impressa, e se '8' não for um dígito (ou seja, se isdigit retornar 0), a string "8 não é um " será impressa.

Biblioteca de tratamento de caracteres

```
1  /* Fig. 8.3: fig08_03.c
2     Usando funções islower, isupper, tolower, toupper */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main( void )
7  {
8     printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
9        "De acordo com islower:",
10       islower( 'p' ) ? "p é uma " : "p não é uma ",
11       "letra minúscula",
12       islower( 'P' ) ? "P é uma " : "P não é uma ",
13       "letra minúscula",
14       islower( '5' ) ? "5 é uma " : "5 não é uma ",
15       "letra minúscula",
16       islower( '!' ) ? "! é uma " : "! não é uma ",
17       "letra minúscula" );
18
```

Figura 8.3 ■ Exemplo do uso de `islower`, `isupper`, `tolower` e `toupper`. (Parte 1 de 2.)

Biblioteca de tratamento de caracteres

```
19      printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
20         "De acordo com isupper:",
21         isupper( 'D' ) ? "D é uma " : "D não é uma ",
22         "letra maiúscula",
23         isupper( 'd' ) ? "d é uma " : "d não é uma ",
24         "letra maiúscula",
25         isupper( '8' ) ? "8 é uma " : "8 não é uma ",
26         "letra maiúscula",
27         isupper( '$' ) ? "$ é uma " : "$ não é uma ",
28         "letra maiúscula" );
29
30      printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31         "u convertido em maiúscula é ", toupper( 'u' ),
32         "7 convertido em maiúscula é ", toupper( '7' ),
33         "$ convertido em maiúscula é ", toupper( '$' ),
34         "L convertido em minúscula é ", tolower( 'L' ) );
35      return 0; /* indica conclusão bem-sucedida */
36   } /* fim do main */
```

Biblioteca de tratamento de caracteres

```
De acordo com islower:
p é uma letra minúscula
P não é uma letra minúscula
5 não é uma letra minúscula
! não é uma letra minúscula

De acordo com isupper:
D é uma letra maiúscula
d não é uma letra maiúscula
8 não é uma letra maiúscula
$ não é uma letra maiúscula

u convertido em maiúscula é U
7 convertido em maiúscula é 7
$ convertido em maiúscula é $
L convertido em minúscula é l
```

Figura 8.3 ■ Exemplo do uso de `islower`, `isupper`, `tolower` e `toupper`. (Parte 2 de 2.)

Biblioteca de tratamento de caracteres

- A Figura 8.3 demonstra as funções islower, isupper, tolower e toupper.
- A função islower determina se seu argumento é uma letra minúscula (a–z).
- A função isupper determina se seu argumento é uma letra maiúscula (A–Z).
- A função tolower converte uma letra maiúscula em minúscula, e retorna a letra minúscula.

Biblioteca de tratamento de caracteres

- Se o argumento não for uma letra maiúscula, tolower retornará o argumento inalterado.
- A função toupper converte uma letra minúscula em maiúscula, e retorna a letra maiúscula.
- Se o argumento não for uma letra minúscula, toupper retornará o argumento inalterado.

Biblioteca de tratamento de caracteres

```c
/* Fig. 8.4: fig08_04.c
   Usando funções isspace, iscntrl, ispunct, isprint, isgraph */
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
      "De acordo com isspace:",
      "Newline", isspace( '\n' ) ? " é um " : " não é um ",
      "caractere de espaço em branco", "Tabulação horizontal",
      isspace( '\t' ) ? " é um " : " não é um ",
      "caractere de espaço em branco",
      isspace( '%' ) ? "% é um " : "% não é um ",
      "caractere de espaço em branco" );

   printf( "%s\n%s%s%s\n%s%s\n\n", "De acordo com iscntrl:",
      "Newline", iscntrl( '\n' ) ? " é um " : " não é um ",
      "caractere de controle", iscntrl( '$' ) ? "$ é um " :
      "$ não é um ", "caractere de controle" );

   printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
      "De acordo com ispunct:",
      ispunct( ';' ) ? "; é um " : "; não é um ",
      "caractere de pontuação",
      ispunct( 'Y' ) ? "Y é um " : "Y não é um ",
      "caractere de pontuação",
      ispunct( '#' ) ? "# é um " : "# não é um ",
      "caractere de pontuação" );

```

Biblioteca de tratamento de caracteres

```
31      printf( "%s\n%s%s\n%s%s%s\n\n", "De acordo com isprint:",
32         isprint( '$' ) ? "$ é um " : "$ não é um ",
33         "caractere imprimível",
34         "Alert", isprint( '\a' ) ? " é um " : " não é um ",
35         "caractere imprimível" );
36
37      printf( "%s\n%s%s\n%s%s%s\n", "De acordo com isgraph:",
38         isgraph( 'Q' ) ? "Q é um " : "Q não é um ",
39         "caractere imprimível diferente de um espaço",
40         "Space", isgraph( ' ' ) ? " é um " : " não é um ",
41         "caractere imprimível diferente de um espaço" );
42      return 0; /* indica conclusão bem-sucedida */
43   } /* fim do main */
```

```
De acordo com isspace:
Newline é um caractere de espaço em branco
Tabulação horizontal é um caractere de espaço em branco
% não é um caractere de espaço em branco

De acordo com iscntrl:
Newline é um caractere de controle
$ não é um caractere de controle

De acordo com ispunct:
; é um caractere de pontuação
Y não é um caractere de pontuação
# é um caractere de pontuação
```

Figura 8.4 ■ Exemplo do uso de `isspace`, `iscntrl`, `ispunct`, `isprint` e `isgraph`. (Parte 1 de 2.)

Biblioteca de tratamento de caracteres

```
De acordo com isprint:
$ é um caractere imprimível
Alert não é um caractere imprimível

De acordo com isgraph:
Q é um caractere imprimível diferente de espaço
Space não é um caractere imprimível diferente de espaço
```

Figura 8.4 ■ Exemplo do uso de `isspace`, `iscntrl`, `ispunct`, `isprint` e `isgraph`. (Parte 2 de 2.)

Biblioteca de tratamento de caracteres

- A Figura 8.4 demonstra as funções isspace, iscntrl, ispunct, isprint e isgraph.
- A função isspace determina se um caractere é um dos caracteres de espaço em branco a seguir: espaço (' '), form feed ('\f'), newline ('\n'), carriage return ('\r'), tabulação horizontal ('\t') ou tabulação vertical ('\v').

Biblioteca de tratamento de caracteres

- A função iscntrl determina se um caractere é um dos caracteres de controle a seguir: tabulação horizontal ('\t'), tabulação vertical ('\v'), form feed ('\f'), alert ('\a'), backspace ('\b'), carriage return ('\r') ou newline ('\n').
- A função ispunct determina se um caractere é um caractere imprimível diferente de um espaço, de um dígito ou de uma letra, por exemplo $, #, (,), [,], {, }, ;, : ou %.
- A função isprint determina se um caractere pode ser exibido na tela (incluindo o caractere de espaço).
- A função isgraph é a mesma que isprint, mas ela não inclui o caractere de espaço.

Funções de conversão de strings

Protótipo da função	Descrição da função
`double atof( const char *nPtr );`	Converte a string `nPtr` em `double`.
`int atoi( const char *nPtr );`	Converte a string `nPtr` em `int`.
`long atol( const char *nPtr );`	Converte a string `nPtr` em `long int`.
`double strtod( const char *nPtr, char **endPtr );`	Converte a string `nPtr` em `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converte a string `nPtr` em `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converte a string `nPtr` em `unsigned long`.

Figura 8.5 ■ Funções de conversão de strings da biblioteca de utilitários gerais.

Funções de conversão de strings

- Esta seção apresenta as funções de conversão de strings da biblioteca de utilitários gerais(<stdlib.h>).
- Essas funções convertem strings de dígitos em valores inteiros e de ponto flutuante.
- A Figura 8.5 resume as funções de conversão de strings.
- Observe que const é usado para declarar a variável nPtr nos cabeçalhos de função (leia da direita para a esquerda, já que "nPtr é um ponteiro para uma constante de caractere"); const determina que o valor do argumento não será modificado.

Funções de conversão de strings

```c
/* Fig. 8.6: fig08_06.c
   Usando atof */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   double d; /* variável para manter a string convertida */

   d = atof( "99.0" );

   printf( "%s%.3f\n%s%.3f\n",
      "A string \"99.0\" convertida em double é ", d,
      "O valor convertido dividido por 2 é ", d / 2.0 );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
A string "99.0" convertida em double é 99.000
O valor convertido dividido por 2 é 49.500
```

Figura 8.6 ■ Exemplo do uso de atof.

Funções de conversão de strings

- A função atof (Figura 8.6) converte seu argumento — uma string que representa um número em ponto flutuante — em um valor double.
- A função retorna o valor double.
- Se o valor convertido não puder ser representado — por exemplo, se o primeiro caractere da string for uma letra —, o comportamento da função atof será indefinido.

Funções de conversão de strings

```
1  /* Fig. 8.7: fig08_07.c
2     Usando atoi */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     int i; /* variável para manter string convertida */
9
10    i = atoi( "2593" );
11
12    printf( "%s%d\n%s%d\n",
13       "A string \"2593\" convertida em int é ", i,
14       "O valor convertido menos 593 é ", i - 593 );
15    return 0; /* indica conclusão bem-sucedida */
16 } /* fim do main */
```

```
A string "2593" convertida em int é 2593
O valor convertido menos 593 é 2000
```

Figura 8.7 ■ Exemplo do uso de atoi.

Funções de conversão de strings

- A função atoi (Figura 8.7) converte seu argumento — uma string de dígitos que representa um inteiro — em um valor int.
- A função retorna o valor int.
- Se o valor convertido não puder ser representado, o comportamento da função atoi.

Funções de conversão de strings

```
1  /* Fig. 8.8: fig08_08.c
2     Usando atol */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     long l; /* variável para manter a string convertida */
9
10    l = atol( "1000000" );
11
12    printf( "%s%ld\n%s%ld\n",
13       "A string \"1000000\" convertida em long int é ", l,
14       "O valor convertido dividido por 2 é ", l / 2 );
15    return 0; /* indica conclusão bem-sucedida */
16 } /* fim do main */
```

```
A string "1000000" convertida em long int é 1000000
O valor convertido dividido por 2 é 500000
```

Figura 8.8 ■ Exemplo do uso de atol.

Funções de conversão de strings

- A função atol (Fig. 8.8) (Figura 8.8) converte seu argumento — uma string de dígitos que representa um inteiro long — em um valor long.
- A função retorna o valor long.
- Se o valor convertido não puder ser representado, o comportamento da função atol será indefinido.
- Se int e long forem armazenados em 4 bytes, a função atoi e a função atol funcionarão de modo idêntico.

Funções de conversão de strings

```c
/* Fig. 8.9: fig08_09.c
   Usando strtod */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   /* inicializa ponteiro de string */
   const char *string = "51.2% são admitidos"; /* inicializa string */

   double d; /* variável para manter sequência convertida */
   char *stringPtr; /* cria ponteiro char */

   d = strtod( string, &stringPtr );

   printf( "A string \"%s\" é convertida em\n", string );
   printf( "valor double %.2f e a string \"%s\"\n", d, stringPtr );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
A string "51.2% são admitidos" é convertida no valor
double 51.20 e na string "% são admitidos"
```

Figura 8.9 ■ Exemplo do uso de strtod.

Funções de conversão de strings

- A função strtod (Figura 8.9) converte uma sequência de caracteres que representam um valor de ponto flutuante em double.
- A função recebe dois argumentos — uma string (char *) e um ponteiro de uma string (char **).
- A string contém a sequência de caracteres a ser convertida.
- O ponteiro recebe o endereço do lugar ocupado pelo 1º caractere que vem imediatamente após a parte convertida da string. A linha 14
  - d = strtod(string, &stringPtr);
- indica que d recebe o valor double convertido de string, e que stringPtr recebe o local do primeiro caractere após o valor convertido (51.2) na string.

Funções de conversão de strings

```
1  /* Fig. 8.10: fig08_10.c
2     Usando strtol */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     const char *string = "-1234567abc"; /* inicializa ponteiro de string */
9
10    char *remainderPtr; /* cria ponteiro de char */
11    long x; /* variável para manter sequência convertida */
12
13    x = strtol( string, &remainderPtr, 0 );
14
15    printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
16       "A string original é ", string,
17       "O valor convertido é ", x,
18       "O resto da string original é ",
19       remainderPtr,
20       "O valor convertido mais 567 é ", x + 567 );
21    return 0; /* indica conclusão bem-sucedida */
22 } /* fim do main */
```

```
A string original é "-1234567abc"
O valor convertido é -1234567
O resto da string original é "abc"
O valor convertido mais 567 é -1234000
```

Figura 8.10 ■ Exemplo do uso de strtol.

Funções de conversão de strings

- A função strtol (Figura 8.10) converte em long uma sequência de caracteres que representa um inteiro.
- A função recebe três argumentos — uma string (char *), um ponteiro para uma string e um inteiro.
- A string contém a sequência de caracteres a ser convertida.
- O ponteiro recebe o endereço do lugar ocupado pelo 1º caractere que vem imediatamente após a parte convertida da string.
- O inteiro especifica a base do valor sendo convertido.

Funções de conversão de strings

- A linha 13
  - x = strtol(string, &remainderPtr, 0);

  indica que x recebe o valor long convertido a partir de string.
- O segundo argumento, remainderPtr, recebe o restante da string após a conversão.
- O uso de NULL no segundo argumento faz com que o restante da string seja ignorado.
- O terceiro argumento, 0, indica que o valor a ser convertido pode estar em formato octal (base 8), decimal (base 10) ou hexadecimal (base 16).

Funções de conversão de strings

- A base pode ser especificada como 0 ou como qualquer valor entre 2 e 36.
- Veja no Apêndice C: Sistemas numéricos uma explicação detalhada dos sistemas de numeração octal, decimal e hexadecimal.
- As representações numéricas dos inteiros de base 11 a 36 utilizam os caracteres A-Z para representar os valores de 10 a 35.
- Por exemplo, os valores hexadecimais consistem nos dígitos 0-9 e nos caracteres A-F.
- Um inteiro de base 36 pode consistir nos dígitos 0-9 e nos caracteres A-Z.

Funções de conversão de strings

```
1  /* Fig. 8.11: fig08_11.c
2     Usando strtoul */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main( void )
7  {
8     const char *string = "1234567abc"; /* inicializa ponteiro de string */
9     unsigned long x; /* variável para manter sequência convertida  */
10    char *remainderPtr; /* cria ponteiro de char */
11
12    x = strtoul( string, &remainderPtr, 0 );
13
14    printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
15       "A string original é ", string,
16       "O valor convertido é ", x,
17       "O resto da string original é ",
18       remainderPtr,
19       "O valor convertido menos 567 é ", x - 567 );
20    return 0; /* indica conclusão bem-sucedida */
21 } /* fim do main */
```

```
A string original é "1234567abc"
O valor convertido é 1234567
O resto da string original é "abc"
O valor convertido menos 567 é 1234000
```

Figura 8.11 ■ Exemplo do uso de strtoul.

Funções de conversão de strings

- A função strtoul (Figura 8.11) converte em unsigned long uma sequência de caracteres que representam um inteiro unsigned long.
- função funciona de forma idêntica à função strtol.
- A instrução
  - x = strtoul(string, &remainderPtr, 0);

  na Figura 8.11 indica que x recebe o valor unsigned long convertido de string.
- O segundo argumento, &remainderPtr, recebe o restante da string após a conversão.
- O terceiro argumento, 0, indica que o valor a ser convertido pode estar em formato octal, decimal ou hexadecimal.

Funções da biblioteca-padrão de entrada/saída

Protótipo da função	Descrição da função
`int getchar( void );`	Insere o caractere seguinte da entrada-padrão e o retorna como um inteiro.
`char *fgets( char *s, int n, FILE *stream);`	Insere caracteres do fluxo especificado para o array `s` até que um caractere de newline ou de fim de arquivo seja encontrado, ou até que `n - 1` bytes sejam lidos. Neste capítulo, especificamos o fluxo como `stdin` — o fluxo de entrada-padrão, que normalmente é usado na leitura de caracteres do teclado. Um caractere nulo de finalização é anexado ao array. Retorna a string que foi lida em `s`.
`int putchar( int c );`	Imprime o caractere armazenado em `c` e o retorna como um inteiro.
`int puts( const char *s );`	Imprime a string `s` seguida por um caractere de newline. Retorna um inteiro diferente de zero se for bem-sucedida, ou EOF, se ocorrer um erro.
`int sprintf( char *s, const char *format, ... );`	Equivalente a `printf`, exceto que a saída é armazenada no array `s` em vez de impressa na tela. Retorna o número de caracteres escritos em `s`, ou EOF, se ocorrer um erro.
`int sscanf( char *s, const char *format, ... );`	Equivalente a `scanf`, exceto que a entrada é lida a partir do array `s` em vez do teclado. Retorna o número de itens lidos com sucesso pela função, ou EOF, se ocorrer um erro.

Figura 8.12 ■ Funções de caractere e de string da biblioteca-padrão de entrada/saída.

Funções da biblioteca-padrão de entrada/saída

- Esta seção apresenta diversas funções da biblioteca-padrão de entrada/saída (<stdio.h>) que, especificamente, manipulam dados de caractere e de string.
- A Figura 8.12 resume as funções de entrada/saída de caractere e de string da biblioteca-padrão de entrada/saída.

Funções da biblioteca-padrão de entrada/saída

```
1  /* Fig. 8.13: fig08_13.c
2     Usando gets e putchar */
3  #include <stdio.h>
4
5  void reverse( const char * const sPtr ); /* protótipo */
6
7  int main( void )
8  {
9     char sentence[ 80 ]; /* cria array de char */
10
11    printf( "Digite uma linha de texto:\n" );
12
13    /* usa fgets para ler linha de texto */
```

Figura 8.13 ■ Exemplo do uso de fgets e putchar. (Parte 1 de 2.)

Funções da biblioteca-padrão de entrada/saída

```
14      fgets( sentence, 80, stdin );
15
16      printf( "\nA linha impressa na ordem inversa é:\n" );
17      reverse( sentence );
18      return 0; /* indica conclusão bem-sucedida */
19   } /* fim do main */
20
21   /* envia caracteres recursivamente na string na ordem reversa */
22   void reverse( const char * const sPtr )
23   {
24      /* se final da string */
25      if ( sPtr[ 0 ] == '\0' ) { /* caso básico */
26         return;
27      } /* fim do if */
28      else { /* se não for final da string */
29         reverse( &sPtr[ 1 ] ); /* etapa de recursão */
30         putchar( sPtr[ 0 ] ); /* usa putchar para exibir caractere */
31      } /* fim do else */
32   } /* fim da função reverse */
```

```
Digite uma linha de texto:
Caracteres e Strings

A linha impressa na ordem inversa é:
sgnirtS e sretcarahC
```

```
Digite uma linha de texto:
apos a sopa

A linha impressa ao contrário é:
apos a sopa
```

Figura 8.13 ■ Exemplo do uso de fgets e putchar. (Parte 2 de 2.)

Funções da biblioteca-padrão de entrada/saída

- A Figura 8.13 usa as funções fgets e putchar ler uma linha de texto da entrada-padrão (teclado) e enviar, recursivamente, os caracteres da linha na ordem inversa.
- A função fgets lê caracteres da entrada-padrão em seu primeiro argumento — um array de chars —, até que um indicador de newline ou de fim de arquivo seja encontrado, ou até que o número máximo de caracteres seja lido.
- O número máximo de caracteres é um a menos do que o valor especificado no segundo argumento de fgets.

Funções da biblioteca-padrão de entrada/saída

- O terceiro argumento especifica o fluxo do qual os caracteres são lidos — nesse caso, usamos o fluxo de entrada-padrão (stdin).
- Um caractere nulo ('\0') é anexado ao array ao final da leitura.
- A função putchar imprime seu argumento de caractere.
- O programa chama a função recursiva reverse para imprimir a linha de texto na ordem inversa.
- Se o primeiro caractere do array recebido por reverse for o caractere nulo'\0', reverse retorna.

Funções da biblioteca-padrão de entrada/saída

- Caso contrário, reverse é chamado novamente com o endereço do subarray que começa no elemento s[1], e o caractere s[0] é enviado com putchar quando a chamada recursiva é concluída.
- A ordem das duas instruções na parte else da estrutura if faz com que reverse caminhe até o caractere nulo que indica o fim da string antes que um caractere seja impresso.
- Quando as chamadas recursivas são concluídas, os caracteres são enviados na ordem reversa.

Funções da biblioteca-padrão de entrada/saída

```
1  /* Fig. 8.14: fig08_14.c
2     Usando getchar e puts */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char c; /* variável para manter caractere digitado pelo usuário */
8     char sentence[ 80 ]; /* cria array de char */
9     int i = 0; /* inicializa contador i */
10
11    /* pede ao usuário que digite linha de texto */
12    puts( "Digite uma linha de texto:" );
13
```

Figura 8.14 ■ Exemplo do uso de getchar e puts. (Parte 1 de 2.)

Funções da biblioteca-padrão de entrada/saída

```
14      /* usa getchar para ler cada caractere */
15      while ( ( c = getchar() ) != '\n') {
16      sentence[ i++ ] = c;
17      } /* end while */
18
19      sentence[ i ] = '\0'; /* finaliza string */
20
21      /* usa puts para exibir a sentença */
22      puts( "\nA linha digitada foi:" );
23      puts( sentence );
24      return 0; /* indica conclusão bem-sucedida */
25   } /* fim do main */
```

```
Digite uma linha de texto:
Isso é um teste.

A linha digitada foi:
Isso é um teste.
```

Figura 8.14 ■ Exemplo do uso de getchar e puts. (Parte 2 de 2.)

Funções da biblioteca-padrão de entrada/saída

- A Figura 8.14 usa as funções getchar e puts para ler caracteres da entrada-padrão no array de caracteres sentence, e imprimir o array de caracteres como uma string.
- A função getchar lê um caractere da entrada-padrão e retorna o caractere como um inteiro.
- A função puts recebe uma string (char *) como um argumento e imprime a string seguida por um caractere de newline.
- O programa para de inserir caracteres quando getchar lê o caractere de newline inserido pelo usuário para finalizar a linha de texto.
- Um caractere nulo é anexado ao array sentence (linha 19), de modo que o array possa ser tratado como uma string.
- Depois, a função puts imprime a string contida em sentence.

Funções da biblioteca-padrão de entrada/saída

```c
/* Fig. 8.15: fig08_15.c
   Usando sprintf */
#include <stdio.h>

int main( void )
{
   char s[ 80 ]; /* cria array de char */
   int x; /* valor x a ser inserido */
   double y; /* valor y a ser inserido */

   printf( "Digite um inteiro e um double:\n" );
   scanf( "%d%lf", &x, &y );

   sprintf( s, "inteiro:%6d\ndouble:%8.2f", x, y );

   printf( "%s\n%s\n",
      "A saída formatada armazenada no array s é:", s );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Digite um inteiro e um double:
298 87.375
A saída formatada armazenada no array s é:
inteiro: 298
double:   87.38
```

Figura 8.15 ■ Exemplo do uso de sprintf.

Funções da biblioteca-padrão de entrada/saída

- A Figura 8.15 usa a função sprintf para imprimir os dados formatados no array s — array de caracteres.
- A função usa os mesmos especificadores de conversão de printf (veja no Capítulo 9 uma discussão detalhada sobre formatação).
- O programa solicita que um valor int e um valor double sejam formatados e impressos no array s.
- O array s é o primeiro argumento de sprintf.

Funções da biblioteca-padrão de entrada/saída

```
1  /* Fig. 8.16: fig08_16.c
2     Usando sscanf */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char s[] = "31298 87.375"; /* inicializa array s */
8     int x; /* valor x a ser inserido */
9     double y; /* valor y a ser inserido */
10
11    sscanf( s, "%d%lf", &x, &y );
12    printf( "%s\n%s%6d\n%s%8.3f\n",
13            "Os valores armazenados no array de caracteres s são:",
14            "integer:", x, "double:", y );
15    return 0; /* indica conclusão bem-sucedida */
16 } /* fim do main */
```

```
Os valores armazenados no array de caracteres s são:
integer:  31298
double:   87.375
```

Figura 8.16 ■ Exemplo do uso de sscanf.

Funções da biblioteca-padrão de entrada/saída

- A Figura 8.16 usa a função sscanf para ler dados formatados do array de caracteres s.
- A função usa os mesmos especificadores de conversão de scanf.
- O programa lê um int e um double do array s e armazena os valores em x e y, respectivamente.
- Os valores de x e y são impressos.
- O array s é o primeiro argumento de sscanf.

Funções de manipulação de strings da biblioteca de tratamento de strings

Protótipo da função	Descrição da função
`char *strcpy( char *s1, const char *s2 )`	Copia string s2 no array s1. O valor de s1 é retornado.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copia no máximo n caracteres da string s2 no array s1. O valor de s1 é retornado.
`char *strcat( char *s1, const char *s2 )`	Acrescenta a string s2 ao array s1. O primeiro caractere de s2 sobrescreve o caractere nulo de finalização de s1. O valor de s1 é retornado.
`char *strncat( char *s1, const char *s2, size_t n )`	Acrescenta no máximo n caracteres da string s2 ao array s1. O primeiro caractere de s2 sobrescreve o caractere nulo de finalização de s1. O valor de s1 é retornado.

Figura 8.17 ■ Funções de manipulação de strings da biblioteca de tratamento de strings.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A biblioteca de tratamento de strings (<string.h>) oferece muitas funções úteis de manipulação de dados de string (cópia e concatenação de strings), comparação de strings, pesquisa de caracteres e outras strings dentro de strings, separação de strings em tokens (partes lógicas) e determinação do comprimento das strings.
- Esta seção apresenta as funções de manipulação da biblioteca de tratamento de strings.
- As funções estão resumidas na Figura 8.17.
- Todas as funções — exceto strncpy — têm o caractere nulo acrescido ao seu resultado.

Funções de manipulação de strings da biblioteca de tratamento de strings

- As funções strncpy e strncat especificam um parâmetro do tipo size_t, que é um tipo definido pelo padrão C como o tipo inteiro do valor retornado pelo operador sizeof.

Funções de manipulação de strings da biblioteca de tratamento de strings

Dica de portabilidade 8.2

O tipo size_t *é um sinônimo dependente do sistema para os tipos* unsigned long *e* unsigned int.

Dica de prevenção de erro 8.2

Ao usar funções da biblioteca de tratamento de strings, inclua o cabeçalho <string.h>.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strcpy copia seu segundo argumento (uma string) em seu primeiro argumento (um array de caracteres que deve ser grande o suficiente para armazenar a string e seu caractere nulo de finalização, que também é copiado).
- A função strncpy é equivalente a strcpy, exceto que strncpy especifica o número de caracteres a serem copiados da string para o array.
- A função strncpy não copia, necessariamente, o caractere nulo de finalização de seu segundo argumento.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Um caractere de término de string é escrito somente se os caracteres a serem copiados tiverem, pelo menos, um caractere a mais que a string.
- Por exemplo, se "test" for o segundo argumento, um caractere nulo de finalização é escrito somente se o terceiro argumento de strncpy tiver, pelo menos, 5 caracteres (quatro caracteres em "test" somado a um caractere nulo de finalização).
- Se o terceiro argumento for maior que 5, caracteres nulos serão acrescentados ao array até que o número total de caracteres, especificado pelo terceiro argumento, seja escrito.

Funções de manipulação de strings da biblioteca de tratamento de strings

Erro comum de programação 8.5
Não acrescentar um caractere nulo de finalização ao primeiro argumento de um strncpy *quando o terceiro argumento*
for menor ou igual ao comprimento da string no segundo argumento.

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.18: fig08_18.c
   Usando strcpy e strncpy */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char x[] = "Parabéns a você"; /* inicializa array de char x */
   char y[ 25 ]; /* cria array de char y */
   char z[ 15 ]; /* cria array de char z */

   /* copia conteúdo de x em y */
   printf( "%s%s\n%s%s\n",
      "A string no array x é: ", x,
      "A string no array y é: ", strcpy( y, x ) );

   /* copia primeiros 14 caracteres de x em z.
      Não copia caractere nulo. */
   strncpy( z, x, 14 );

   z[ 14 ] = '\0'; /* termina string em z */
   printf( "A string no array z é: %s\n", z );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
A string no array x é: Parabéns a você
A string no array y é: Parabéns a você
A string no array z é: Parabéns a
```

Figura 8.18 ■ Exemplo do uso de strcpy e strncpy.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A Figura 8.18 usa strcpy para copiar a string inteira do array x no array y, e usa strncpy para copiar os primeiros 14 caracteres do array x no array z.
- Um caractere nulo ('\0') é acrescentado ao array z, pois a chamada a strncpy no programa não escreve um caractere nulo de finalização (o terceiro argumento é menor em tamanho do que a string do segundo argumento).

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.19: fig08_19.c
   Usando strcat e strncat */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 20 ] = "Feliz "; /* inicializa array de char s1 */
   char s2[] = "Ano Novo "; /* inicializa array de char s2 */
   char s3[ 40 ] = ""; /* inicializa array de char s3 como vazio */

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );

   /* concatena s2 com s1 */
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );

   /* concatena 6 primeiros caracteres de s1 com s3. Coloca '\0'
      após último caractere */
   printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );

   /* concatena s1 com s3 */
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
s1 = Feliz
s2 = Ano Novo
strcat( s1, s2 ) = Feliz Ano Novo
strncat( s3, s1, 6 ) = Feliz
strcat( s3, s1 ) = Feliz Feliz Ano Novo
```

Figura 8.19 ■ Exemplo do uso de strcat e strncat.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strcat acrescenta seu segundo argumento (uma string) a seu primeiro argumento (um array de caracteres que contém uma string).
- O primeiro caractere do segundo argumento substitui o nulo ('\0'), que indica o fim da string no primeiro argumento.
- Você precisa garantir que o array usado para armazenar a primeira string seja grande o suficiente para armazenar a primeira e a segunda strings, além do caractere nulo de finalização copiado da segunda string.
- A função strncat acrescenta um número especificado de caracteres da segunda à primeira string.
- Um caractere nulo de finalização é automaticamente acrescentado ao resultado.
- A Figura 8.19 demonstra as funções strcat e strncat.

Funções de manipulação de strings da biblioteca de tratamento de strings

Protótipo da função	Descrição da função
`int strcmp( const char *s1, const char *s2 );`	
	Compara a string `s1` com a string `s2`. A função retorna 0, menor do que 0 ou maior do que 0 se `s1` for igual, menor ou maior do que `s2`, respectivamente.
`int strncmp( const char *s1, const char *s2, size_t n );`	
	Compara até `n` caracteres da string `s1` com a string `s2`. A função retorna 0, menor do que 0 ou maior do que 0 se `s1` for igual, menor ou maior do que `s2`, respectivamente.

Figura 8.20 ■ Funções de comparação da biblioteca de tratamento de strings.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Esta seção apresenta as funções de comparação de string da biblioteca de tratamento de strings, **strcmp** e **strncmp**.
- A Figura 8.20 contém seus protótipos e uma breve descrição de cada função.

Funções de manipulação de strings da biblioteca de tratamento de strings

```
1  /* Fig. 8.21: fig08_21.c
2     Usando strcmp e strncmp */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     const char *s1 = "Feliz Ano Novo"; /* inicializa ponteiro char */
9     const char *s2 = "Feliz Ano Novo"; /* inicializa ponteiro char */
10    const char *s3 = "Boas Férias"; /* inicializa ponteiro char */
11
12    printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
13       "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
14       "strcmp(s1, s2) = ", strcmp( s1, s2 ),
15       "strcmp(s1, s3) = ", strcmp( s1, s3 ),
16       "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
17
18    printf("%s%2d\n%s%2d\n%s%2d\n",
19       "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
20       "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
21       "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
22    return 0; /* indica conclusão bem-sucedida */
23 } /* fim do main */
```

Funções de manipulação de strings da biblioteca de tratamento de strings

```
s1 = Feliz Ano Novo
s2 = Feliz Ano Novo
s3 = Boas Férias

strcmp(s1, s2) = 0
strcmp(s1, s3) = 1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) = 0
strncmp(s1, s3, 7) = 6
strncmp(s3, s1, 7) = -6
```

Figura 8.21 ■ Exemplo do uso de `strcmp` e `strncmp`.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A Figura 8.21 compara três strings usando strcmp e strncmp.
- A função strcmp compara seu primeiro argumento de string com seu segundo argumento de string, caractere por caractere.
- A função retorna 0 se as strings forem iguais, um valor negativo se a primeira string for menor do que a segunda, e um valor positivo se a primeira string for maior que a segunda.
- A função strncmp tem as mesmas características de strcmp, exceto que strncmp compara até certo número especificado de caracteres.
- Em uma string, a função strncmp não compara caracteres após um caractere nulo.
- O programa imprime o valor inteiro retornado por cada chamada de função.

Funções de manipulação de strings da biblioteca de tratamento de strings

Erro comum de programação 8.6

Pressupor que strcmp *e* strncmp *retornem 1 quando seus argumentos são iguais consiste em um erro lógico. Ambas as funções retornam 0 (curiosamente, o equivalente do valor falso da C) quando seus argumentos são iguais. Portanto, ao testar a igualdade de duas strings, o resultado da função* strcmp *ou* strncmp *deve ser comparado a 0 para determinar se as strings são iguais.*

Funções de manipulação de strings da biblioteca de tratamento de strings

- Para entender o que significa exatamente uma string ser ‘maior’ ou ‘menor’ que outra string, considere o processo de ordenação alfabética de uma série de sobrenomes.
- O leitor, sem dúvida, coloca ‘Nunes’ antes de ‘Silva’, pois, no alfabeto, a primeira letra de ‘Nunes’ vem antes da primeira letra de ‘Silva’.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Mas o alfabeto é mais que uma lista de 26 letras: é uma lista ordenada de caracteres.
- Cada letra ocupa uma posição específica dentro da lista. 'Z' é mais que simplesmente uma letra do alfabeto; 'Z' é, mais especificamente, a 26ª letra do alfabeto.
- Como é que o computador sabe qual letra vem antes e qual vem depois?
- Todos os caracteres são representados dentro do computador como códigos numéricos; quando o computador compara duas strings, ele, na realidade, compara dois códigos numéricos dos caracteres nas strings.

Funções de manipulação de strings da biblioteca de tratamento de strings

Dica de portabilidade 8.3

Os códigos numéricos internos usados na representação de caracteres podem ser diferentes; isso depende de cada computador.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Em um esforço para padronizar as representações de caracteres, a maioria dos fabricantes de computador projetou suas máquinas para que elas utilizassem um de dois esquemas de codificação populares — ASCII ou EBCDIC.
- ASCII significa 'American Standard Code for Information Interchange', e EBCDIF significa 'Extended Binary Coded Decimal Interchange Code'.
- Existem outros esquemas de codificação, mas esses são os mais populares.
- O padrão Unicode® esboça uma especificação para produzir uma codificação consistente da grande maioria dos caracteres e símbolos do mundo.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Para saber mais sobre o Unicode, visite www.unicode.org.
- ASCII, EBCDIC e Unicode são chamados de conjuntos de caracteres.
- Na realidade, manipulações de strings e caracteres envolvem a manipulação dos códigos numéricos apropriados, e não dos caracteres propriamente ditos.
- Isso explica a interoperabilidade de caracteres e pequenos inteiros em C.
- Como é significativo dizer que um código numérico é maior, menor ou igual a outro código numérico, torna-se possível ligar diversos caracteres ou strings uns aos outros referindo-se aos códigos dos caracteres.
- O Apêndice B lista os códigos de caracteres ASCII.

Funções de manipulação de strings da biblioteca de tratamento de strings

Protótipo e descrição da função
`char *strchr( const char *s, int c );`
Localiza a primeira ocorrência do caractere c na string s. Se c for encontrado, um ponteiro para c em s é retornado. Caso contrário, um ponteiro NULL é retornado.
`size_t strcspn( const char *s1, const char *s2 );`
Determina e retorna o tamanho do segmento inicial da string s1 que consiste em caracteres *não* contidos na string s2.
`size_t strspn( const char *s1, const char *s2 );`
Determina e retorna o tamanho do segmento inicial da string s1 que consiste apenas em caracteres contidos na string s2.
`char *strpbrk( const char *s1, const char *s2 );`
Localiza a primeira ocorrência na string s1 de qualquer caractere na string s2. Se um caractere da string s2 for encontrado, um ponteiro para o caractere na string s1 é retornado. Caso contrário, um ponteiro NULL é retornado.
`char *strrchr( const char *s, int c );`
Localiza a última ocorrência de c na string s. Se c for encontrado, um ponteiro para c na string s é retornado. Caso contrário, um ponteiro NULL é retornado.
`char *strstr( const char *s1, const char *s2 );`
Localiza a primeira ocorrência na string s1 da string s2. Se a string for encontrada, um ponteiro para a string em s1 é retornado. Caso contrário, um ponteiro NULL é retornado.
`char *strtok( char *s1, const char *s2 );`
Uma sequência de chamadas para strtok separa a string s1 em 'tokens' — partes lógicas, por exemplo, palavras em uma linha de texto — separados por caracteres contidos na string s2. A primeira chamada contém s1 como primeiro argumento, e para que as chamadas seguintes continuem a separar tokens na mesma string, elas deverão conter NULL como primeiro argumento. Um ponteiro para o token em vigor é retornado por cada chamada. Se não houver mais tokens quando a função for chamada, NULL será retornado.

Figura 8.22 ■ Funções de manipulação de strings da biblioteca de tratamento de strings.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Esta seção apresenta as funções da biblioteca de tratamento de strings usadas na pesquisa de strings de caracteres e outras strings.
- As funções estão resumidas na Figura 8.22.
- As funções strcspn e strspn retorna size_t.

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.23: fig08_23.c
   Usando strchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string = "Isso é um teste"; /* inicializa ponteiro de char */
   char character1 = 'u'; /* inicializa character1 */
   char character2 = 'z'; /* inicializa character2 */

   /* se character1 foi achado na string */
   if ( strchr( string, character1 ) != NULL ) {
      printf( "\'%c\' foi achado em \"%s\".\n",
         character1, string );
   } /* fim do if */
   else { /* se character1 não foi achado */
      printf( "\'%c\' não foi achado em \"%s\".\n",
         character1, string );
   } /* fim do else */

   /* se character2 foi achado na string */
   if ( strchr( string, character2 ) != NULL ) {
      printf( "\'%c\' foi achado em \"%s\".\n",
         character2, string );
   } /* fim do if */
   else { /* se character2 não foi achado */
      printf( "\'%c\' não foi achado em \"%s\".\n",
         character2, string );
   } /* fim do else */

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

```
'u' foi encontrado em "Isso é um teste".
'z' não foi encontrado em "Isso é um teste".
```

Figura 8.23 ■ Exemplo do uso de strchr.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strchr procura pela primeira ocorrência de um caractere em uma string.
- Se o caractere for encontrado, strchr retornará
- um ponteiro para o caractere na string; caso contrário, strchr retornará NULL.
- A Figura 8.23 usa strchr para procurar a primeira ocorrência de 'a' e 'z' na string "Isso é um teste".

Funções de manipulação de strings da biblioteca de tratamento de strings

```
1  /* Fig. 8.24: fig08_24.c
2     Usando strcspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8     /* inicializa dois ponteiros char */
9     const char *string1 = "O valor é 3.14159";
10    const char *string2 = "1234567890";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
13       "string1 = ", string1, "string2 = ", string2,
14       "O comprimento do segmento inicial de string1",
```

Figura 8.24 ■ Exemplo do uso de strcspn. (Parte 1 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

```
15          "que não contém caracteres de string2 = ",
16          strcspn( string1, string2 ) );
17       return 0; /* indica conclusão bem-sucedida */
18    } /* fim do main */
```

```
string1 = O valor é 3.14159
string2 = 1234567890

O comprimento do segmento inicial de string1
que não contém caracteres de string2 = 13
```

Figura 8.24 ■ Exemplo do uso de strcspn. (Parte 2 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strcspn (Figura 8.24) determina o comprimento da parte inicial da string em seu primeiro argumento que não contém nenhum caractere da string em seu segundo argumento.
- A função retorna o comprimento do segmento.

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.25: fig08_25.c
   Usando strpbrk */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "Isso é um teste"; /* inicializa ponteiro de char */
   const char *string2 = "cerrado"; /* inicializa ponteiro de char */

   printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
      "Dos caracteres em ", string2,
      *strpbrk( string1, string2 ),
      " aparece primeiro em ", string1 );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Dos caracteres em "cerrado"
'o' aparece primeiro em
"Isso é um teste"
```

Figura 8.25 ■ Exemplo do uso de strpbrk.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strpbrk procura, em seu primeiro argumento de string, a primeira ocorrência de qualquer caractere contido em seu segundo argumento de string.
- Se um caractere do segundo argumento for encontrado, strpbrk retornará um ponteiro para o caractere no primeiro argumento; caso contrário, strpbrk retornará NULL.
- A Figura 8.25 mostra um programa que localiza a primeira ocorrência na string1 de qualquer caractere de string2.

Funções de manipulação de strings da biblioteca de tratamento de strings

```
1  /* Fig. 8.26: fig08_26.c
2     Usando strrchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
```

Figura 8.26 ■ Exemplo do uso de `strrchr`. (Parte 1 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

```
 8     /* inicializa ponteiro de char */
 9     const char *string1 = "Em um zoológico encontram-se muitos animais, inclusive zebras";
10
11     int c = 'z'; /* caractere a ser procurado */
12
13     printf( "%s\n%s'%c'%s\"%s\"\n",
14        "O restante de string1 que começa com a",
15        "última ocorrência do caractere ", c,
16        " é: ", strrchr( string1, c ) );
17     return 0; /* indica conclusão bem-sucedida */
18  } /* fim do main */
```

```
O restante da string1 que começa com a
última ocorrência do caractere 'z' é: "zebras"
```

Figura 8.26 ■ Exemplo do uso de strrchr. (Parte 2 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strrchr procura a última ocorrência do caractere especificado em uma string.
- Se o caractere for encontrado, strrchr retornará um ponteiro para o caractere na string; caso contrário, strrchr retornará NULL.
- A Figura 8.26 mostra um programa que procura a última ocorrência do caractere 'z' na string "Em um zoológico encontram-se muitos animais, inclusive zebras."

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.27: fig08_27.c
   Usando strspn */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* inicializa dois ponteiros de char */
   const char *string1 = "O valor é 3.14159";
   const char *string2 = "aéor lsOuv";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
      "string1 = ", string1, "string2 = ", string2,
      "O comprimento do segmento inicial de string1",
      "que contém apenas caracteres da string2 = ",
      strspn( string1, string2 ) );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
string1 = O valor é 3.14159
string2 = aéor lsOuv

O comprimento do segmento inicial da string1
que contém apenas caracteres de string2 = 13
```

Figura 8.27 ■ Exemplo do uso de strspn.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strspn (Figura 8.27) determina o comprimento da parte inicial da string em seu primeiro argumento, que contém apenas caracteres da string em seu segundo argumento.
- A função retorna o comprimento do segmento.

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.28: fig08_28.c
Using strstr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "abcdefabcdef"; /* string a procurar */
   const char *string2 = "def"; /* string a ser procurada */

   printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
      "string1 = ", string1, "string2 = ", string2,
      "O restante de string1 que começa com a",
      "primeira ocorrência de string2 é: ",
      strstr( string1, string2 ) );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
string1 = abcdefabcdef
string2 = def
O restante de string1 que começa com a
primeira ocorrência de string2 é: defabcdef
```

Figura 8.28 ■ Exemplo do uso de strstr.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strstr procura pela primeira ocorrência de seu segundo argumento de string em seu primeiro argumento de string.
- Se a segunda string for encontrada na primeira string, um ponteiro para o local da string no primeiro argumento é retornado.
- A Figura 8.28 usa strstr para encontrar a string "def" na string "abcdefabcdef".

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.29: fig08_29.c
   Usando strtok */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* inicializa o array string */
   char string[] = "Essa é uma sentença com 7 tokens";
   char *tokenPtr; /* cria ponteiro char */

   printf( "%s\n%s\n\n%s\n",
      "A string a ser separada em tokens é:", string,
      "Os tokens são:" );

   tokenPtr = strtok( string, " " ); /* inicia a separação em tokens */

   /* continua a separar até que tokenPtr se transforme em NULL */
   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " ); /* obtêm próximo token */
   } /* fim do while */

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

Figura 8.29 ■ Exemplo do uso de strtok. (Parte 1 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

```
A string a ser separada em tokens é:
Essa é uma sentença com 7 tokens

Os tokens são:
Essa
é
uma
sentença
com
7
tokens
```

Figura 8.29 ■ Exemplo do uso de `strtok`. (Parte 2 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função strtok (Figura 8.29) é usada para separar uma string em uma série de tokens.
- Um 'token' é uma sequência de caracteres separados por delimitadores (normalmente consistem em espaços ou marcas de pontuação, mas podem ser qualquer caractere).
- Por exemplo, em uma linha de texto, cada palavra pode ser considerada um token, e os espaços e sinais de pontuação que separam as palavras podem ser considerados delimitadores.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Para separar uma string em tokens — ou seja, para dividi-la em tokens (supondo que a string contenha mais de um token) —, várias chamadas a strtok são necessárias.
- A primeira chamada a strtok contém dois argumentos: uma string a ser separada e uma string que contém os caracteres que separam os tokens.
- Na Figura 8.29, a instrução statement
  - `/* inicia separação em tokens */`
    `tokenPtr = strtok( string, " " );`

  atribui a tokenPtr um ponteiro para o primeiro token em string.

Funções de manipulação de strings da biblioteca de tratamento de strings

- O segundo argumento, " ", indica que os tokens são separados por espaços.
- A função strtok procura o primeiro caractere em string que não seja um caractere delimitador (espaço).
- Isso inicia o primeiro token.
- A função, então, encontra o caractere delimitador seguinte na string e o substitui por um caractere nulo ('\0') para finalizar o token atual.
- A função strtok reserva um ponteiro para o caractere que vier depois do token na string, e retorna um ponteiro para o token em corrente.

Funções de manipulação de strings da biblioteca de tratamento de strings

- As chamadas subsequentes a strtok na linha 21 continuam a separar a string.
- Essas chamadas têm NULL como seu primeiro argumento.
- O argumento NULL indica que a chamada a strtok deve continuar separando os tokens a partir do local em string reservado pela última chamada a strtok.
- Se não restar nenhum token quando strtok for chamada, strtok retornará NULL.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Você pode alterar a string delimitadora a cada nova chamada a strtok.
- A Figura 8.29 usa strtok para separar em tokens a string "Esta é uma sentença com 7 tokens".
- Cada token é impresso separadamente.
- A função strtok modifica a string de entrada ao colocar \0 no final de cada token; portanto, deve ser feita uma cópia da string se ela tiver que ser usada novamente no programa após a chamada a strtok.

Funções de manipulação de strings da biblioteca de tratamento de strings

Protótipo da função	Descrição da função
`void *memcpy( void *s1, const void *s2, size_t n );`	Copia n caracteres do objeto apontado por s2 no objeto apontado por s1. Um ponteiro para o objeto resultante é retornado.
`void *memmove( void *s1, const void *s2, size_t n );`	Copia n caracteres do objeto apontado por s2 no objeto apontado por s1. A cópia é realizada como se os caracteres fossem copiados primeiro do objeto apontado por s2 em um array temporário, e depois do array temporário no objeto apontado por s1. Um ponteiro para o objeto resultante é retornado.
`int memcmp( const void *s1, const void *s2, size_t n );`	Compara os primeiros n caracteres dos objetos apontados por s1 e s2. A função retorna 0, menor ou maior do que 0 se s1 for igual, menor ou maior que s2, respectivamente.
`void *memchr( const void *s, int c, size_t n );`	Localiza a primeira ocorrência de c (convertida em unsigned char) nos primeiros n caracteres do objeto apontado por s. Se c for encontrado, um ponteiro para c no objeto será retornado. Caso contrário, NULL será retornado.
`void *memset( void *s, int c, size_t n );`	Copia c (convertido em unsigned char) nos primeiros n caracteres do objeto apontado por s. Um ponteiro para o resultado é retornado.

Figura 8.30 ■ Funções de memória da biblioteca de tratamento de strings.

Funções de manipulação de strings da biblioteca de tratamento de strings

- As funções da biblioteca de tratamento de strings apresentadas nesta seção manipulam, comparam e procuram blocos de memória.
- As funções tratam os blocos de memória como arrays de caracteres, e podem manipular qualquer bloco de dados.
- A Figura 8.30 resume as funções de memória da biblioteca de tratamento de strings.
- Nas discussões de função, 'objeto' se refere a um bloco de dados.

Funções de manipulação de strings da biblioteca de tratamento de strings

- Os parâmetros de ponteiro para essas funções são declarados void *, de modo que podem ser usados para manipular a memória em qualquer tipo de dado.
- Vimos que um ponteiro para qualquer tipo de dado pode ser atribuído diretamente a um ponteiro do tipo void *, e um ponteiro do tipo void * pode ser atribuído diretamente a um ponteiro de qualquer tipo de dado.
- Por esse motivo, essas funções podem receber ponteiros para qualquer tipo de dado.
- Como um ponteiro void * não pode ser desreferenciado, cada função recebe um argumento de tamanho que especifica o número de caracteres (bytes) que a função processará.
- Para simplificar, os exemplos nesta seção manipulam arrays de caracteres (blocos de caracteres).

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.31: fig08_31.c
   Usando memcpy */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 18 ]; /* cria array de char s1 */
   char s2[] = "Copie essa string"; /* inicializa array de char s2 */

   memcpy( s1, s2, 18 );
   printf( "%s\n%s\"%s\"\n",
      "Depois que s2 é copiada em s1 com memcpy,",
      "s1 contém ", s1 );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
Depois que s2 é copiada em s1 com memcpy,
s1 contém "Copie essa string"
```

Figura 8.31 ■ Exemplo do uso de memcpy.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função memcpy copia um número especificado de caracteres do objeto apontado por seu segundo argumento no objeto apontado por seu primeiro argumento.
- A função pode receber um ponteiro para qualquer tipo de objeto.
- O resultado dessa função será indefinido se os dois objetos estiverem sobrepostos na memória (ou seja, se eles forem partes do mesmo objeto) — nesse caso, use memmove.
- A Figura 8.31 usa memcpy para copiar a string do array s2 noarray s1.

Funções de manipulação de strings da biblioteca de tratamento de strings

```
1  /* Fig. 8.32: fig08_32.c
2     Usando memmove */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
```

Figura 8.32 ■ Exemplo do uso de memmove. (Parte 1 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

```
 7  {
 8     char x[] = "Lar Doce Lar"; /* inicializa array de char x */
 9
10     printf( "%s%s\n", "A string no array x antes de memmove é: ", x );
11     printf( "%s%s\n", "A string no array x depois de memmove é: ",
12        memmove( x, &x[ 5 ], 10 ) );
13     return 0; /* indica conclusão bem-sucedida */
14  } /* fim do main */
```

```
A string no array x antes de memmove é: Lar Doce Lar
A string no array x depois de memmove é: Lar Doce Lar
```

Figura 8.32 ■ Exemplo do uso de memmove. (Parte 2 de 2.)

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função memmove, assim como memcpy, copia um número especificado de bytes do objeto apontado por seu segundo argumento no objeto apontado por seu primeiro argumento.
- A cópia é feita como se os bytes fossem copiados do segundo argumento em um array de caracteres temporário, depois copiados do array temporário no primeiro argumento.
- Isso permite que os caracteres de uma parte da string sejam copiados em outra parte da mesma string.
- A Figura 8.32 usa memmove para copiar os 10 últimos bytes do array x nos 10 primeiros bytes do array x.

Funções de manipulação de strings da biblioteca de tratamento de strings

Erro comum de programação 8.7
As funções de manipulação de string diferentes de memmove *que copiam caracteres possuem resultados indefinidos*
quando a cópia ocorre entre partes da mesma string.

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.33: fig08_33.c
   Usando memcmp */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[] = "ABCDEFG"; /* inicializa array de char s1 */
   char s2[] = "ABCDXYZ"; /* inicializa array de char s2 */

   printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
      "s1 = ", s1, "s2 = ", s2,
      "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
      "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
      "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp( s1, s2, 4 ) = 0
memcmp( s1, s2, 7 ) = -1
memcmp( s2, s1, 7 ) = 1
```

Figura 8.33 ■ Exemplo do uso de memcmp.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função memcmp (Figura 8.33) compara o número especificado de caracteres de seu primeiro argumento com os caracteres correspondentes de seu segundo argumento.
- A função retorna um valor maior que 0 se o primeiro argumento for maior que o segundo, retorna 0 se os argumentos forem iguais e retorna um valor menor que 0 se o primeiro argumento for menor que o segundo

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.34: fig08_34.c
   Usando memchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *s = "Isso é uma string"; /* inicializa ponteiro de char */

   printf( "%s\'%c\'%s\"%s\"\n",
      "O restante de s após o caractere ", 'r',
      " ser encontrado é ", memchr( s, 'r', 16 ) );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
O restante de s após o caractere 'r' ser encontrado é "ring"
```

Figura 8.34 ■ Exemplo do uso de memchr.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função memchr procura a primeira ocorrência de um byte, representada como unsigned char, no número especificado de bytes de um objeto.
- Se o byte for encontrado, um ponteiro do byte no objeto é retornado; caso contrário, um ponteiro NULL é retornado.
- A Figura 8.34 procura o caractere (byte) 'r' na string "Isso é uma string".

Funções de manipulação de strings da biblioteca de tratamento de strings

```c
/* Fig. 8.35: fig08_35.c
   Usando memset */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char string1[ 15 ] = "BBBBBBBBBBBBBB"; /* inicializa string1 */

   printf( "string1 = %s\n", string1 );
   printf( "string1 depois de memset = %s\n", memset( string1, 'b', 7 ) );
   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

```
string1 = BBBBBBBBBBBBBB
string1 depois de memset = bbbbbbbBBBBBBB
```

Figura 8.35 ■ Exemplo do uso de memset.

Funções de manipulação de strings da biblioteca de tratamento de strings

- A função **memset** copia o valor do byte de seu segundo argumento nos primeiros *n* bytes do objeto apontado por seu primeiro argumento, onde *n* é especificado pelo terceiro argumento.
- A Figura 8.35 usa memset para copiar 'b' nos 7 primeiros bytes de string1.

Funções de manipulação de strings da biblioteca de tratamento de strings

Protótipo da função	Descrição da função
`char *strerror( int errornum );`	
	Mapeia `errornum` em uma string de texto completa de uma maneira específica ao compilador e ao local (por exemplo, a mensagem pode aparecer em diferentes idiomas, com base no seu local). Um ponteiro da string é retornado.
`size_t strlen( const char *s );`	
	Determina o comprimento da string `s`. O número de caracteres anteriores ao caractere nulo de finalização é retornado.

Figura 8.36 ■ Outras funções da biblioteca de tratamento de strings.

Funções de manipulação de strings da biblioteca de tratamento de strings

- As duas funções restantes da biblioteca de tratamento de strings são strerror e strlen.
- A Figura 8.36 resume as funções strerror e strlen.

Outras funções da biblioteca de tratamento de strings

```
 1  /* Fig. 8.37: fig08_37.c
 2     Usando strerror */
 3  #include <stdio.h>
 4  #include <string.h>
 5
 6  int main( void )
 7  {
 8     printf( "%s\n", strerror( 2 ) );
 9     return 0; /* indica conclusão bem-sucedida */
10  } /* fim do main */
```

```
No such file or directory
```

Figura 8.37 ■ Exemplo do uso de strerror.

Outras funções da biblioteca de tratamento de strings

- A função strerror obtém um número de erro e cria uma string de mensagem de erro. Um ponteiro para a string é retornado.
- A Figura 8.37 demonstra strerror.

Outras funções da biblioteca de tratamento de strings

```
1  /* Fig. 8.38: fig08_38.c
2     Usando strlen */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main( void )
7  {
8  /* inicializa 3 ponteiros char */
9     const char *string1 = "abcdefghijklmnopqrstuvwxyz";
10    const char *string2 = "dois";
11    const char *string3 = "Murici";
12
13    printf("%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n",
14       "O comprimento de ", string1, " é ",
15        ( unsigned long ) strlen( string1 ),
16       "O comprimento de ", string2, " é ",
17        ( unsigned long ) strlen( string2 ),
18       "O comprimento de ", string3, " é ",
19        ( unsigned long ) strlen( string3 ) );
20    return 0; /* indica conclusão bem-sucedida */
21 } /* fim do main */
```

```
O comprimento de "abcdefghijklmnopqrstuvwxyz" é 26
O comprimento de "dois" é 4
O comprimento de "Murici" é 6
```

Figura 8.38 ■ Exemplo do uso de strlen.

Outras funções da biblioteca de tratamento de strings

- A função strlen obtém uma string como um argumento e retorna o número de caracteres na string — o caractere nulo de finalização não está incluído no comprimento.
- A Figura 8.38 demonstra a função strlen.

Lembre-se

“ Antes que o software possa ser reutilizável, ele primeiro deve ser utilizável. ” Ralph Johnson