Considerando que o ponteiro `str` seja definido como:

```
const char *str = "bola";
```

A chamada de `F()` a seguir é legal?

```
F(str);
```

Conversão de Strings Numéricos em Números (Seção 8.9)

72. Descreva o funcionamento da função **atoi()**.
73. Em qual situação o resultado retornado por **atoi()** é ambíguo e por quê?
74. Compare a função `StrEmInt()`, apresentada na **Seção 8.10.8**, com a função **atoi()** da biblioteca padrão de C.
75. Descreva o funcionamento da função **strtod()**.
76. (a) Para quer serve o último parâmetro de **strtod()**? (b) Como esse parâmetro deve ser usado?
77. Por que, quando o segundo parâmetro de **strtod()** é **NULL**, o valor retornado por essa função pode ser ambíguo?

8.12 Exercícios de Programação

8.12.1 Fácil

EP8.1) Escreva uma função, denominada `TransformaStr()`, que recebe um string como primeiro parâmetro e um caractere como segundo parâmetro. Quando o segundo parâmetro for o caractere `'M'`, essa função deve transformar o string de tal modo que todas as suas letras passem a ser maiúsculas. Quando o segundo parâmetro for `'m'`, a função deve transformar o string de tal modo que todas as suas letras sejam minúsculas. Quando o segundo parâmetro não for `'M'` ou `'m'` essa função não deve promover nenhuma transformação no string. A função deverá retornar o endereço do string transformado. [**Sugestão**: Utilize as funções **tolower()** e **toupper()** discutidas na **Seção 8.7.2**.]

EP8.2) Escreva um programa, semelhante àquele apresentado como exemplo na **Seção 8.10.9**, que lê e verifica a validade de uma data no formato ISO 8601 (i.e., no formato: `aaaa/mm/dd`). [**Sugestão**: Estude o exemplo apresentado na **Seção 8.10.9** e descubra o que precisa ser alterado naquele programa para obter a solução para o problema corrente.]

EP8.3) Implemente uma função, denominada `ComparaStrings()`, funcionalmente equivalente à função **strcmp()**. [**Sugestão**: Use um laço de repetição para comparar os strings caractere a caractere. Se for encontrada uma diferença entre respectivos caracteres ou for encontrado o caractere terminal de um dos strings encerre o laço e retorne a diferença entre os dois últimos caracteres acessados.]

EP8.4) Implemente uma função, denominada `EncontraPrimeiroChar()`, funcionalmente equivalente à função **strchr()**. [**Sugestão**: Utilize um laço de repetição para comparar cada caractere do string com o caractere procurado. Esse laço deve encerrar quando o caractere procurado ou o caractere terminal for encontrado. No primeiro caso, a função retorna o endereço do caractere,

enquanto que, no segundo caso, ela retorna **NULL**. Lembre-se que o caractere procurado pode ser o caractere terminal do string.]

EP8.5) Implemente uma função, denominada `EncontraUltimoChar()`, funcionalmente equivalente à função **strrchr()**. [**Sugestão**: Faça um ponteiro local à função apontar para o caractere terminal do string usando **strchr()**. Então, use um laço de repetição que decremente esse ponteiro até que ele aponte para o caractere procurado ou seja menor do que o ponteiro que aponta para o início do string. No primeiro caso, a função retorna o valor do ponteiro auxiliar, enquanto que, no segundo caso, ela retorna **NULL**.]

EP8.6) Escreva uma função que retorna `1` quando um string possui apenas letras e dígitos ou `0`, em caso contrário. [**Sugestão**: Utilize a função **isalnum()** discutida na **Seção 8.7.1**.]

EP8.7) Escreva uma função que substitui cada caractere de tabulação de um string por um espaço em branco. [**Sugestão**: Utilize **strchr()** para localizar cada caractere de tabulação `'\t'` e então substitua-o.]

EP8.8) (a) Escreva uma função, denominada `OcorrenciasCar()`, que conta o número de ocorrências de um caractere num string. [**Sugestões**: (1) Use uma variável local para contar o número de ocorrências solicitado. Essa variável deve ser iniciada com zero. (2) Use um laço **while** cuja condição de parada seja o fato de o parâmetro que representa o string apontar para o caractere terminal desse string. (3) No corpo do laço, verifique se esse parâmetro aponta para o caractere procurado e, se for o caso, incremente a variável que armazena o número de ocorrências. Em seguida, incremente o parâmetro. (4) Após o final do laço, retorne o valor da variável que conta as ocorrências.] (b) Escreva um programa que lê strings e caracteres isolados via teclado e informa o número de ocorrências de cada caractere no respectivo string. O programa deve encerrar quando o usuário introduzir um string vazio. [**Sugestão**: Use `LeString()` e `LeCaractere()` da biblioteca LeituraFacil para ler os strings e os caracteres, respectivamente.]

EP8.9) (a) Escreva uma função que substitui todas as ocorrências de um dado caractere num string por outro caractere. O protótipo dessa função deve ser:

```
char *SubstituiCaracteres( char *str, int substituir,
                           int novo )
```

Os parâmetros dessa função são interpretados como:

- `str` é o string no qual serão feitas as substituições
- `substituir` é o caractere que será substituído
- `novo` é o caractere que substituirá as ocorrências do segundo parâmetro

O retorno dessa função deve ser o endereço do string recebido como parâmetro. (b) Escreva um programa para testar a função `SubstituiCaracteres()`. [**Sugestão**: Utilize como base a função `OcorrenciasCar()` solicitada no exercício **EP8.8**. Então, em vez de contar as ocorrências de um dado caractere, você as substituirá.]

EP8.10) Escreva uma função que remove todos caracteres que não são letras de um string. [**Sugestões**: (1) Use a função **isalpha()** discutida na **Seção 8.7.1**. (2) Use as sugestões apresentadas para o exercício **EP8.9**.]

EP8.11) (a) Escreva uma função semelhante a `StrEmInt()`, apresentada na **Seção 8.10.8**, que efetue conversões de strings em qualquer base entre `2` e `10`, e não apenas na base `10`, como faz a função apresentada. (b) Escreva um programa para testar a função solicitada em (a). [**Sugestão**: Acrescente um parâmetro que represente a base de conversão à lista de parâmetros de `StrEmInt()`. Então, substitua os usos de base `10` no corpo da função pelo novo parâmetro.]

EP8.12) (a) Escreva uma função que retorna o token de ordem `n` de um string, se este existir; caso contrário, a função deve retornar **NULL**. O protótipo dessa função deve ser:

```
char *EnesimoToken( char *str, const char *separadores,
                    int n )
```

Nesse protótipo, os parâmetros têm os seguintes significados:

- `str` é o string no qual o enésimo token será procurado
- `separadores` é um string contendo os possíveis separadores de tokens
- `n` é o número de ordem do token desejado

(b) Escreva um programa para testar a função `EnesimoToken()`. [**Sugestão**: Utilize a função **strtok()** para ler e descartar `n - 1` tokens do string. Se **strtok()** retornar **NULL** antes, não existe enésimo token. Caso contrário, chame e retorne o valor retornado por **strtok()**.]

EP8.13) (a) Escreva uma função que copia os `n` caracteres iniciais de um string. O protótipo dessa função deve ser:

```
char *CopiaInicio( char *destino, const char *origem,
                   int n )
```

Os parâmetros dessa função são interpretados como:

- `destino` é o array que receberá a cópia
- `origem` é o string que doará os caracteres
- `n` é o número de caracteres iniciais que serão copiados.

O retorno dessa função deve ser o endereço do array recebido como primeiro parâmetro. (b) Escreva um programa para testar a função `CopiaInicio()`. [**Observação**: A função `CopiaInicio()` é semelhante à função **strncpy()** da biblioteca padrão de C.]

EP8.14) (a) Escreva uma função que copia os `n` caracteres finais de um string. O protótipo dessa função deve ser:

```
char *CopiaFinal( char *destino, const char *origem,
                  int n )
```

Nesse protótipo, os parâmetros têm os seguintes significados:

- `destino` é o array que receberá a cópia
- `origem` é o string que doará os caracteres
- `n` é o número de caracteres finais que serão copiados, sem incluir o caractere terminal do string `origem`.

O retorno dessa função deve ser o endereço do array recebido como parâmetro. (b) Escreva um programa para testar a função `CopiaFinal()`. [**Sugestão**: Utilize um ponteiro `p` local à função e faça-o apontar para o primeiro caractere a ser copiado e, então, chame **strcpy()** como `strcpy(destino, p)`.]

EP8.15) (a) Escreva uma função que copia `n` caracteres de um string a partir de uma dada posição. O protótipo dessa função deve ser:

```
char *CopiaNCaracteres( char *destino, const char *origem,
                        int pos, int n )
```

As interpretações dos parâmetros nesse protótipo são as seguintes:

- `destino` é o array que receberá a cópia
- `origem` é o string que fornecerá os caracteres
- `pos` é a posição do string `origem` a partir da qual os caracteres serão copiados
- `n` é o número de caracteres que serão copiados.

O retorno dessa função deve ser o endereço do array recebido como parâmetro. (b) Escreva um programa para testar a função `CopiaNCaracteres()`. [**Sugestão**: Utilize a função `CopiaInicio()` solicitada no exercício **EP8.13**.]

EP8.16) Escreva um programa que verifica se um número de CPF introduzido pelo usuário é válido. Um número de CPF tem 11 dígitos divididos em duas partes: a parte principal com 9 dígitos e os dígitos verificadores, que são os dois últimos dígitos. A validação de um número de CPF segue o procedimento descrito abaixo:

- Verificação do primeiro dígito de controle (penúltimo dígito do número):
  - ☞ Multiplique os inteiros representados pelos dígitos da parte principal, do primeiro ao último, respectivamente, por `10`, `9`, `8`, ..., `2` e some os resultados obtidos.
  - ☞ Calcule o resto da divisão do resultado obtido no item anterior por `11`.
  - ☞ Se o resto da divisão for `0` ou `1`, o primeiro dígito verificador deverá ser igual a `0`; caso contrário, esse dígito deverá ser igual a `11` menos o referido resto de divisão.
- Verificação do segundo dígito de controle (último dígito do número):
  - ☞ Multiplique os inteiros representados pelos dígitos da parte principal e pelo primeiro dígito de controle, do primeiro ao último, respectivamente, por `11`, `10`, `9`, ..., `2` e some os resultados obtidos.
  - ☞ Calcule o resto da divisão do resultado obtido no item anterior por `11`.
  - ☞ Se o resto da divisão for `0` ou `1`, o segundo dígito verificador deverá ser igual a `0`; caso contrário, esse dígito deverá ser igual a `11` menos o último resto de divisão.

A principal diferença entre essas duas verificações é que a segunda inclui o primeiro dígito de controle.
[**Sugestão**: Esse problema é semelhante àquele de verificação de número de PIS/PASEP apresentado como exemplo na **Seção 8.10.3**.]

EP8.17) Escreva um programa semelhante àquele apresentado na **Seção 8.10.11**, denominado `multiplicaints` (ou `multiplicaints.exe`) que multiplica os argumentos de linha de comando que acompanham o nome do programa se eles forem todos números inteiros.

EP8.18) Escreva um programa semelhante àquele apresentado na **Seção 8.10.11**, denominado `somareais` (ou `somareais.exe`), que soma números reais passados para o programa como argumentos de linha de comando. [**Sugestão**: Use a

função **strtod()**, discutida na **Seção 8.9.2**, para converter os parâmetros do programa em números reais.]

EP8.19) Escreva um programa que recebe um valor inteiro positivo `N` como argumento de linha de comando e apresenta como resultado a sequência de Fibonacci que contém `N` termos. Se `N` não for um valor válido para o programa ou estiver ausente, o programa deve responder adequadamente. [**Sugestão**: Use como modelos os exemplos apresentados nas **Seções 8.10.11** e **6.6.7**.]

EP8.20) Escreva um programa que recebe um valor inteiro positivo `N` como argumento de linha de comando e informa se `N` faz parte de alguma sequência de Fibonacci. Se `N` não for um valor válido para o programa ou estiver ausente, o programa deve responder adequadamente. [**Sugestão**: Siga a sugestão do exercício anterior.]

EP8.21) Escreva um programa que exibe na tela a segunda metade de um string. [**Sugestões**: (1) Calcule a metade do tamanho do string usando **strlen()**. (2) Faça um ponteiro apontar para essa posição. (3) Use esse ponteiro com uma chamada de **printf()**.]

EP8.22) Escreva um programa que exibe na tela a primeira metade de um string. [**Sugestões**: (1) Encontre a posição central do string usando **strlen()**. (2) Use um laço de contagem que chame **putchar()** para exibir cada caractere do string do seu início até a sua posição central.]

EP8.23) (a) Escreva uma função que acrescenta um caractere ao final de um string. O protótipo dessa função deve ser:

```
char *AcrescentaCaractere(char *str, int c, size_t tam)
```

Os parâmetros dessa função são interpretados como:

- `str` é o string que terá um caractere acrescentado
- `c` é o caractere que será acrescentado
- `tam` é o tamanho do array que contém o string.

O retorno dessa função deve ser o endereço do string alterado, se for possível acrescentar o caractere ou **NULL**, se não houver espaço suficiente. (b) Escreva um programa que lê strings e caracteres isolados via teclado, tenta acrescentar cada caractere no respectivo string e informa o resultado da operação. O programa deve encerrar quando o usuário digitar apenas `[ENTER]` quando instado a introduzir um string. [**Sugestões**: (1) Use **strlen()** para checar se há espaço suficiente para acréscimo de um novo caractere. (2) Se houver espaço para acréscimo, use, por exemplo, **strlen()** ou **strchr()** para acessar o local da inserção, que deve ser a posição corrente do caractere `'\0'`. (3) Não esqueça de acrescentar um novo caractere terminal ao string.]

EP8.24) A função `ApresentaErro()`, definida na **Seção 8.10.8**, é repetitiva na indicação de erro. Crie uma nova função, denominada `ApresentaErro2()`, com um parâmetro adicional que represente um string a ser apresentado como mensagem de erro.

EP8.25) Escreva um programa que apresenta na tela uma frase (string) introduzida pelo usuário em forma de escada. Isto é, cada palavra constituinte da frase é

exibida numa linha separada e endentada em relação à palavra anterior, como por exemplo:

```
Isto
    e'
      um
        teste
```

[**Sugestão**: Use a **strtok()** para extrair cada palavra da frase e a função **strlen()** para calcular a endentação de uma palavra em relação àquela exibida na linha anterior.]

EP8.26) (a) Escreva uma função, denominada `InverteString()`, que copia um string invertido (segundo parâmetro) para um array (primeiro parâmetro). (b) Escreva um programa que lê strings via teclado e apresenta-os invertidos na tela. [**Sugestão**: Defina um ponteiro `p` local à função `InverteString()` e faça-o apontar para o último caractere do string usando **strchr()**. Então, use um laço de repetição para copiar cada caractere correntemente por `p` para cada elemento do array e decrementar esse ponteiro. O laço deve encerrar quando `p` apontar para o endereço inicial do string.]

EP8.27) (a) Escreva uma função, denominada `EhVogal()`, que verifica se o caractere recebido como parâmetro é vogal. [**Sugestão**: Use **strchr()** para verificar se o caractere faz parte do string constante `"aeiouAEIOU"`.] (b) Escreva uma função, denominada `EhConsoante()`, que verifica se um caractere é consoante. [**Sugestão**: Use **isalpha()** e `EhVogal()`.] (c) Escreva um programa que lê uma palavra via teclado e informa quantas vogais e consoantes a palavra possui. [**Sugestão**: Use a função `LeNome()`, definida na **Seção 8.10.1** e as funções solicitadas nos itens (a) e (b).]

EP8.28) Escreva um programa que lê um número positivo menor do que `5000` e apresenta na tela o número correspondente usando algarismos romanos. Esse programa deve ser funcionalmente equivalente àquele solicitado no exercício **EP5.18**, mas deve usar os arrays de strings: `unidades[]`, `dezenas[]`, `centenas[]` e `milhares[]` para armazenar os strings constantes que correspondem, respectivamente, à possível unidade, dezena, centena e milhar do número lido.

EP8.29) (a) Escreva uma função, denominada `OcorrenciasStr()`, que conta o número de ocorrências de um string (primeiro parâmetro) em outro string (segundo parâmetro). [**Sugestões**: (1) Defina uma variável de contagem e inicie-a com `0`. (2) Crie um laço de repetição infinito no corpo do qual a função **strstr()** é chamada tendo o string a ser procurado como segundo parâmetro. (3) Na primeira execução do corpo do laço, o primeiro parâmetro de **strstr()** deve ser o primeiro parâmetro da função que está sendo implementada. Nas execuções subsequentes do corpo do laço, esse parâmetro de **strstr()** deve ser acrescido do tamanho do string procurado. (4) Esse laço deve encerrar quando **strstr()** retornar **NULL**.] (b) Escreva um programa para testar a função solicitada no item (a).

EP8.30) **Preâmbulo**: Um **palíndromo** é uma palavra ou frase que pode ser lida tanto em sentido direto quanto ao contrário (i.e., de trás para a frente). Tipicamente,

num palíndromo, não se levam em consideração acentos, pontuações, hifens ou espaços entre palavras. Um dos exemplos de palíndromo mais simples da língua portuguesa é *arara*. **Problema**: Escreva um programa que verifica se dois strings contendo apenas letras e espaços em branco são palíndromos. [**Sugestões**: (1) Use a função `LeNome()`, definida na **Seção 8.10.1**, para ler os dois strings. (2) Remova todos os espaços em branco dos dois strings usando como modelo a função `ComprimeEspacos()` apresentada na **Seção 8.10.13**. (3) Inverta um dos strings usando a função `InverteString()`, solicitada no exercício **EP8.26** e verifique se o string invertido é igual ao outro string. **NB**: Se você adotar essas sugestões, os strings que serão verificados não podem ser constantes.]

EP8.31) **Preâmbulo**: Dois strings constituem **anagramas** se cada um deles é uma combinação dos caracteres do outro. Por exemplo, `"roma"` e `"amor"` são, provavelmente, os anagramas mais conhecidos da língua portuguesa. **Problema**: Escreva um programa que verifica se dois strings introduzidos via linha de comando são anagramas. [**Sugestão**: Use a função `RemoveCaracteresDuplicados()`, definida na **Seção 8.10.14**, para remover caracteres duplicados dos dois string. Então, utilize um laço de repetição para verificar se um string contém todos os caracteres do outro por meio de chamadas de **strchr()**.]

EP8.32) A função `LeNome()`, apresentada na **Seção 8.10.1**, não testa se o string lido é constituído apenas por espaços em branco, de forma que o usuário pode digitar um nome que será invisível quando exibido. Reescreva essa função de maneira a corrigir esse defeito. [**Sugestão**: Use a função `RemoveBrancosInicio()`, definida na **Seção 8.10.7**, para remover eventuais espaços em branco no início do string lido. Então teste se o string se torna vazio após a chamada dessa função. Se esse for o caso, inste o usuário a introduzir um novo nome.]

8.12.2 Moderado

EP8.33) Implemente uma função, denominada `PosicaoEmString()`, funcionalmente equivalente à função **strstr()**.

EP8.34) Escreva um programa que apresenta todos os anagramas (v. exercício **EP8.31**) que podem ser formados com as letras de uma palavra introduzida pelo usuário via teclado. [**Sugestões**: (1) Escreva uma função que lê strings contendo apenas letras. (2) Utilize o método de geração de permutações por ordenação lexicográfica discutido na **Seção 7.11.10** para gerar possíveis anagramas de um string constituído apenas por letras.]

EP8.35) (a) Escreva uma função que recebe como parâmetro um string que representa um número inteiro positivo em base binária e retorna o valor desse número em base decimal. Se for encontrado um caractere do string que não seja `'0'` ou `'1'`, ou se ocorrer overflow, a função solicitada deve usar a função `ApresentaErro()`, definida na **Seção 8.10.8**, para indicar o erro e retornar `-1`. [**Sugestões**: (1) Use um laço **for** para acessar cada caractere do string a partir do último caractere. Esse laço deve encerrar quando todos os caracteres forem levados em consideração ou for encontrado um caractere que não é `'0'` nem `'1'`. (2) Use uma variável para acumular o resultado da conversão

e outra para armazenar o valor da potência de `2` pela qual o valor do dígito corrente será multiplicado. Inicie a primeira variável com `0` e a segunda variável com `1`. (3) Para cada caractere válido encontrado, converta-o em inteiro (v. **Seção 8.10.8**), multiplique-o pela variável que representa a potência de `2` correspondente e acrescente o valor resultante à variável que acumula o resultado da conversão. (4) Verifique se ocorreu overflow como resultado da última atualização do valor convertido (v. **Seção 4.12.7**). Se for o caso, apresente o erro e retorne `-1`. (5) Atualize a variável que armazena a potência.] (b) Escreva um programa que lê um string que supostamente representa um número inteiro positivo em base binária e, se não ocorrer erro, apresenta seu valor em base decimal.

EP8.36) (a) Escreva uma função que remove de um string todas as ocorrências de um dado caractere. O protótipo dessa função deve ser:

```
char *RemoveCaractere(char *str, int remover)
```

Nesse protótipo, `str` é o string que será eventualmente modificado, `remover` é o caractere a ser removido e o retorno da função deve ser o endereço inicial do string. [**Sugestões**: (1) Use dois ponteiros locais à função, denominados `p` e `inicio` e faça-os apontar para string `str` recebido como parâmetro. (2) Use um laço **while** que encerre quando `str` apontar para o caractere terminal do string. No corpo desse laço, copie para o endereço apontado por `p` qualquer caractere que não seja igual ao caractere que será removido e faça `p` e `str` apontarem para um caractere adiante. (3) Depois do laço, acrescente um caractere terminal após o último caractere apontado por `p`. (4) Retorne o valor do ponteiro `inicio`.] (b) Escreva um programa que lê um string e um caractere via teclado e substitui todas as ocorrências do caractere no string. O string deve ser apresentado na tela antes e depois das eventuais substituições.

EP8.37) (a) Implemente uma função, cujo protótipo é:

```
char *RemoveCaracteres(char *str, const char *aRemover)
```

que remove do primeiro string recebido como parâmetro todos os caracteres presentes no segundo parâmetro, que também é um string. O retorno dessa função deve ser o endereço do string eventualmente alterado. (b) Escreva uma função **main()** que recebe dois strings como argumentos de linha de comando e remove do primeiro string todos os caracteres presentes no segundo string. [**Sugestão**: Use a função `RemoveCaractere()` solicitada no exercício **EP8.36**.]

EP8.38) (a) Escreva uma função que remove todas as ocorrências de um dado string em outro string e retorna o número de remoções efetuadas. [**Sugestões**: (1) defina três variáveis: `p`, usada como ponteiro auxiliar; `tamSubstring`, que armazenará o tamanho do substring que será removido e `nRemocoes`, que armazenará o número de remoções. (2) Calcule o tamanho do substring que será removido e atribua-o a `tamSubstring`. (3) Use um laço **while** que encerra quando o parâmetro que representa o string é **NULL** ou aponta para o caractere `'\0'`. (4) No corpo desse laço, use **strstr()** para encontrar a próxima ocorrência do substring no string e atribua o retorno dessa função

a `p`. (5) Se `p` for **NULL**, encerre o laço. Caso contrário, copie para o array apontado por `p` o string que começa em `p + tamSubstring` e incremente a variável `nRemocoes`. (6) Ainda no corpo do laço, atribua `p` ao parâmetro que representa o string.] (b) Escreva um programa que lê dois strings via teclado, remove as ocorrências do segundo string no primeiro e apresenta o resultado da operação.

EP8.39) (a) Escreva uma função que substitui todas as ocorrências de um substring num string por outro substring. [**Sugestões**: (1) Use **strlen()** para calcular os tamanhos dos dois substrings. (2) Use um laço de repetição que encerra quando a função **strstr()** indicar que não há mais ocorrências do substring a ser substituído. (3) No corpo desse laço, determine o espaço para o qual serão copiados os caracteres substitutos, discriminando as substituições em duas categorias, dependendo dos tamanhos dos dois substrings. Isto é, se o tamanho do substring que será substituído for maior do que o daquele que o substituirá, devem-se mover caracteres para trás; caso contrário, devem-se mover caracteres para frente. Nos dois casos, o deslocamento de caracteres deve ser igual à diferença de tamanho entre os dois substrings. (4) Copie os caracteres do substring para o espaço determinado no passo anterior.] (b) Escreva um programa para testar a função especificada em (a). [**Sugestão**: Use a função `OcorrenciasStr()`, solicitada no exercício **EP8.29**, para calcular o tamanho que o string resultante das substituições terá quando a operação estiver concluída e determinar se o array que armazenará o resultado terá espaço suficiente para contê-lo.]

EP8.40) (a) Escreva uma função, denominada `IntEmString()`, que converte um valor do tipo **int** em string. [**Sugestões**: (1) Defina um array de duração fixa local à função para armazenar o resultado da operação. O tamanho desse array é o valor de uma constante simbólica do programa. É vital que o referido array tenha duração fixa, pois, caso contrário, ele seria considerado um zumbi (v. **Seção 7.9.4**). (2) Verifique se o número a ser convertido é negativo e, se for o caso, armazene essa informação numa variável local e considere o valor absoluto do número para conversão. Ao final da conversão, se o número for negativo, o sinal de menos será acrescentado ao string. (3) Para evitar que o string que conterá os dígitos que compõem o número precise ser invertido ao final do processo, armazene-os do final para o início do array. Portanto, o primeiro passo para obter o resultado desejado é armazenar o caractere terminal do string na última posição do array. (4) Use um laço **do-while** para extrair e armazenar no array cada dígito que compõe o número. Enquanto isso é efetuado, conte quantos caracteres estão sendo armazenados e compare esse valor com o tamanho do array para evitar corrupção de memória; i.e., se a quantidade de caracteres (dígitos, sinal e caractere terminal) exceder a capacidade de armazenamento do array, retorne **NULL**, indicando que a conversão não foi bem sucedida. (5) Finalmente, se o número for negativo e ainda houver espaço no array, acrescente o sinal de menos ao string que contém o resultado e retorne o endereço inicial do string (e não do array que o armazena).] (b) Escreva um programa para testar a função especificada no item (a).

EP8.41) (a) Escreva uma função, denominada `DoubleEmString()`, que converte um valor do tipo **double** em string. Essa função deve truncar a parte fracionária na segunda casa decimal. [**Sugestões**: (1) Separe o número em partes inteira e fracionária, conforme ensinado na **Seção 6.5**. (2) Armazene as partes inteira e fracionária no array que conterá o resultado como faz a função `IntEmString()` solicitada no exercício **EP8.40**. (3) Não esqueça que existe um ponto decimal separando as duas partes.] (b) Escreva um programa que testa a função especificada no item (a).

EP8.42) (a) Escreva uma função que separa um string em tokens, como faz a função **strtok()** da biblioteca padrão de C. (b) Escreva um programa que testa a função solicitada no item (a) e compara os resultados obtidos por meio dessa função com aqueles obtidos via **strtok()**. [**Sugestões para o item (a)**: (1) Use um ponteiro de duração fixa local à função, denominado `proximoToken`, para armazenar o endereço do primeiro caractere do próximo token do string recebido como parâmetro. Defina ainda os seguintes ponteiros locais: `s`, que apontará para o string no qual a busca pelo token será efetuada, e `inicio`, que guarda o início do token corrente. (2) Quando o primeiro parâmetro da função em discussão não for **NULL**, a busca pelo próximo token começa no endereço indicado por esse parâmetro. Caso contrário, a busca pelo próximo token começa no endereço armazenado na variável `proximoToken`, a não ser que essa variável também seja **NULL**. Nesse último caso, não há mais token a ser encontrado e a função retorna **NULL**. (3) Quando há possíveis tokens a serem encontrados, saltam-se eventuais separadores (especificados no segundo parâmetro) que se encontrem no início do string no qual a busca será realizada. Se, durante essa operação, o final do string for atingido, não há mais token no string sendo processado e a função retorna **NULL**. Ainda nesse caso, a variável `proximoToken` recebe o valor **NULL**, de forma que a próxima chamada da função tendo **NULL** como primeiro parâmetro não procurará um novo token. (4) Se, após saltar os separadores iniciais, o final do string não for atingido, haverá pelo menos mais um token no string. Então, guarda-se o endereço desse token que será retornado na variável `inicio` e procura-se o final desse token (i.e., um separador de token especificado no segundo parâmetro ou `'\0'`) usando a variável `s`. Quando um separador é encontrado, ele é substituído pelo caractere terminal `'\0'` e à variável `proximoToken` é atribuído o endereço do caractere que segue esse separador. Se, nesse passo não for encontrado nenhum separador, não haverá mais token na próxima chamada da função, e, assim, à variável `proximoToken` é atribuído **NULL**. (5) A última instrução da função retorna o endereço do token encontrado, que foi armazenado na variável `inicio`.