APÊNDICE B

A BIBLIOTECA LEITURAFACIL

B.1 Introdução

Este apêndice serve como referência para a biblioteca LeituraFacil, que foi descrita na **Seção 3.13.2** e usada em inúmeros exemplos de programação apresentados neste livro. Aqui, serão apresentadas todas as implementações de funções que essa biblioteca disponibiliza para uso externo (funções globais), bem como daquelas que auxiliam a implementação da própria biblioteca (funções locais). Para entender bem essas implementações, é essencial que você tenha estudado o **Capítulo 11**, especialmente a **Seção 11.9**.

Este apêndice apresenta ainda a motivação que deu origem à biblioteca LeituraFacil e mostra como compilá-la e integrá-la nos sistemas operacionais e ferramentas de desenvolvimento suportados por este livro.

B.2 Justificativa

Programação se aprende construindo programas interativos e esses programas, normalmente, requerem leitura de dados via teclado. Infelizmente, é justamente aí que reside a maior fonte de frustração e dificuldade encontrada pelo iniciante em programação em C. Para ilustrar essa afirmação, observe, na **Tabela B–1**, o conhecimento necessário para um programador escrever uma instrução para leitura de um único número inteiro em base decimal por meio do teclado usando a biblioteca padrão de C.

Para ler um número inteiro via teclado é preciso saber...	Problemas Potenciais
Que o nome da função a ser chamada para executar essa tarefa é *scanf*.	Convenhamos, ***scanf*** não é um nome nem um pouco mnemônico para quem fala Português. Portanto, o aprendiz terá certa dificuldade para memorizar esse nome.
O que são strings de formatação e como eles devem ser usados com **scanf()**.	Frequentemente, iniciantes confundem strings de formatação de **scanf()** com aqueles usados com **printf()**.

Tabela B–1: Conhecimento Necessário para Leitura de um Número Inteiro

Para ler um número inteiro via teclado é preciso saber...	Problemas Potenciais
Que o especificador de formato a ser usado é %d.	Novamente, o iniciante precisa contar com a memória para não usar um especificador de formato inadequado. Ainda pior se lhe for ensinado, equivocadamente, que ele pode usar %i em substituição a %d.
Que é necessário usar uma variável do tipo int precedida pelo operador de endereço &.	Esquecer-se de preceder com & a variável na qual o valor lido será armazenado é um dos erros mais frequentemente cometidos por iniciantes. Esse erro é sério e traz graves consequências para um programa.

Tabela B–1: Conhecimento Necessário para Leitura de um Número Inteiro

Será justo com o iniciante em programação requerer que ele domine tanto conhecimento para ler um singelo número inteiro?

Agora, suponha que o aprendiz é dedicado e absorve bem todo o conhecimento mencionado na tabela acima. Infelizmente, ele logo descobrirá que seu esforço foi em vão porque, com todo esse conhecimento adquirido, seus programas não atenderão dois critérios de qualidade: robustez e condescendência com o usuário. Quer dizer, se o usuário cometer algum engano na introdução de dados, ele não terá chance de corrigir seu erro e o programa, incapaz de reconhecer o erro, terá um comportamento indefinido. Assim, o iniciante descobrirá que mais conhecimento é necessário para tornar seus programas robustos e amigáveis. Ou seja, como foi visto no **Capítulo 11**, ele precisará acrescentar um *pouco* mais de conhecimento a seu repertório. Mais precisamente, ele precisará ainda aprender:

- Que a função **scanf()** retorna um valor que indica quando uma operação de leitura é bem sucedida.
- Como usar um laço de repetição em conjunto com o valor retornado por **scanf()** para dar novas chances ao usuário quando ele introduz um valor inesperado pelo programa.
- Que utilizando o meio de entrada padrão, apenas caracteres são lidos.
- Que o meio de entrada padrão usa um buffer associado a ele no qual os caracteres digitados são armazenados.
- Como remover caracteres eventualmente remanescentes nesse buffer.

Como se pode depreender das considerações acima, explicar em detalhes o funcionamento da leitura de dados via teclado consome muito tempo e desvia a atenção daquilo que deve ser o enfoque de um texto introdutório de programação. Por isso, neste livro, o funcionamento da entrada de dados via teclado é discutido em detalhes apenas no **Capítulo 11**, quando o aprendiz deverá sentir-se mais confortável com o uso da linguagem C. Antes desse capítulo, os detalhes de operações de leitura de dados via teclado são ocultados, enquanto que, nesse capítulo, a apresentação desse tópico se torna apropriada, pois o foco desse capítulo é exatamente entrada e saída de dados.

A biblioteca LeituraFacil foi a solução encontrada pelo autor deste livro para o ensino de disciplinas introdutórias de programação usando a linguagem C. [A abordagem adotada na criação de LeituraFacil foi inspirada numa concepção do professor Eric Roberts da Universidade de Stanford, EUA (v. **Bibliografia**).]

As funções dessa biblioteca permitem que um iniciante escreva programas robustos e amigáveis que efetuam leitura de dados via teclado de modo tão simples como ocorre numa linguagem algorítmica. Por exemplo, para ler um número inteiro em base decimal e armazená-lo numa variável **x** do tipo **int** sem ter que defrontar com

todo o conhecimento descrito acima, o aprendiz pode chamar a função `LeInteiro()` como:

```
x = LeInteiro();
```

A Tabela B–2 mostra a correspondência entre as funções da biblioteca LeituraFacil e funções do módulo stdio da biblioteca padrão de C. Mas, essas correspondências não significam equivalência funcional entre funções das duas bibliotecas.

Função da Biblioteca Leitura Fácil	Função da Biblioteca Padrão
`LeCaractere()`	getchar()
`LeOpcao()`	getchar()
`LeInteiro()`	scanf() com especificador %d
`LeReal()`	scanf() com especificador %lf
`LeString()`	fgets()

Tabela B–2: Correspondências entre Funções de LeituraFacil e stdio

O uso mais indicado para a biblioteca LeituraFacil consiste em instalá-la, conforme será mostrado adiante e chamar suas funções normalmente, como tem sido visto em inúmeros exemplos neste livro. Mas, existem opções menos práticas de uso:

- Tornar o arquivo `leitura.c` componente de um programa multiarquivo. Além do incômodo de transformar programas monoarquivos simples em programas multiarquivos, esse uso não é recomendado num curso introdutório, pois, nesse caso, programas multiarquivos não devem constituir um tópico essencial de discussão.
- Copiar em seu programa-fonte as definições das funções que ele utiliza. Apesar de essa opção ser melhor do que a opção anterior, ela ainda é um tanto incômoda. Nesse caso, devem ser incluídas no programa não apenas as funções globais que ele precisa usar, como também as funções locais que essas funções globais chamam (v. Seção B.3).

B.3 Implementação

A biblioteca LeituraFacil é constituída por um único módulo dividido em dois arquivos:

- **Arquivo de cabeçalho**. Nesse arquivo encontram-se alusões das funções disponibilizadas pela biblioteca LeituraFacil para uso em qualquer programa (desde que a biblioteca tenha sido devidamente instalada, obviamente).
- **Arquivo de programa**. No contexto de programação multiarquivo, um arquivo de programa usa essa denominação porque ele gera código binário. Essa denominação se contrapõe a *arquivo de cabeçalho*, que não deve gerar código binário. No arquivo de programa da biblioteca LeituraFacil, são definidas as funções disponibilizadas para uso irrestrito (funções com escopo global) e outras funções que auxiliam a implementação da biblioteca (funções com escopo de arquivo).

B.3.1 Cabeçalho

O único arquivo de cabeçalho da biblioteca LeituraFacil, `leitura.h`, contém alusões das funções globais disponibilizadas pela biblioteca para uso em qualquer programa. O conteúdo desse arquivo é apresentado a seguir.

```
/****
 *
```

```
 * Título: LeituraFacil
 *
 * Autor: Ulysses de Oliveira
 *
 * Data de Criação: 22/07/2012
 * Última modificação: 22/07/2012
 *
 * Descrição: Definição do módulo da biblioteca LeituraFacil
 *
 ****/

#ifndef _Leitura_H_
#define _Leitura_H_

extern int LeCaractere(void);
extern int LeInteiro(void);
extern double LeReal(void);
extern int LeString(char *ar, int nElementos);
extern int LeOpcao(const char *opcoes);

#endif
```

As diretivas de pré-processamento **#ifndef** e **#endif**, que aparecem no arquivo de cabeçalho, não são discutidas neste livro pois não fazem parte de seu escopo, mas elas são indispensáveis em arquivos de cabeçalho para prevenir que eles sejam eventualmente incluídos múltiplas vezes. No caso específico desse arquivo de cabeçalho, que só contém alusões de funções, as chances de ocorrer inclusão múltipla são ínfimas e, se isso acontecer, não causará nenhum dano a um programa. Mas, incluir essas diretivas em arquivos de cabeçalho é sempre uma boa norma de programação.

B.3.2 Funções Locais

As funções descritas nesta seção auxiliam a implementação das funções globais que efetivamente fazem parte da biblioteca LeituraFacil. Isso significa que as funções descritas aqui são locais ao arquivo de programa da biblioteca e não podem ser usadas fora dele. Por exemplo, a função `LimpaBuffer()` é essencial na implementação de todas as funções da biblioteca em discussão, mas ela não é necessária em programas que utilizam as funções disponibilizadas por essa biblioteca. Por isso, decidiu-se não tornar global a função `LimpaBuffer()`.

Todas as funções locais são qualificadas com **static** (v. **Seção 5.12.2**) em suas definições e é exatamente esse qualificador que as tornam funções locais. Se fosse desejado tornar pública alguma dessas funções seriam necessárias as seguintes alterações:

- Remoção de **static** na definição da função.
- Inclusão de uma alusão da função no cabeçalho da biblioteca.

LimpaBuffer()

A função `LimpaBuffer()` remove caracteres remanescentes no buffer associado ao meio de entrada padrão após uma leitura de dados. Todas as funções globais da biblioteca LeituraFacil chamam essa função.

```
/****
 *
 * LimpaBuffer(): Limpa o buffer associado ao meio de entrada padrão (stdin)
 *
 * Parâmetros: Nenhum
 *
 * Retorno: O número de caracteres descartados, com exceção do caractere '\n'
```

```
 *
 ****/
static int LimpaBuffer(void)
{
   int carLido, /* Armazena cada caractere lido */
       nCarLidos = 0; /* Conta o número de caracteres lidos */

      /* Lê e descarta cada caractere lido até    */
      /* encontrar '\n' ou getchar() retornar EOF */
   do {
      carLido = getchar(); /* Lê um caractere */
      ++nCarLidos; /* Mais um caractere foi lido */
   } while ((carLido != '\n') && (carLido != EOF));

      /* O último caractere lido foi '\n' ou  */
      /* EOF e não deve ser considerado sobra */
   return nCarLidos - 1;
}
```

Análise: A função `LimpaBuffer()` é semelhante à função `LimpaBuffer2()` discutida na **Seção 10.9.5** e não requer discussão adicional.

MostraMensagem()

A função `MostraMensagem()` é utilizada pelas funções globais da biblioteca LeituraFacil para apresentação de mensagens seguindo um padrão consistente.

```
/****
 *
 * MostraMensagem(): Apresenta uma mensagem na tela com embelezamento
 *
 * Parâmetros: msg (entrada) - mensagem a ser apresentada
 *
 * Retorno: Nada
 *
 * Observação: O objetivo desta função é uniformizar as mensagens emitidas
 *             pelas funções globais da biblioteca LeituraFacil
 *
 ****/
static void MostraMensagem(const char *msg)
{
   printf("\a\n\t>>> %s", msg);
}
```

Análise: A implementação da função `MostraMensagem()` é simples demais e não requer comentários adicionais.

InformaDescarte()

A função `InformaDescarte()` é utilizada pelas funções `LeCaractere()`, `LeInteiro()` e `LeReal()` para apresentação de mensagens relativas a caracteres descartados (i.e., caracteres que restaram no buffer associado à entrada de dados padrão após uma leitura de dados nesse meio de entrada).

```
/****
 *
 * InformaDescarte(): Apresenta uma mensagem na tela informando
 *                    quantos caracteres foram descartados
 *
 * Parâmetros: n (entrada) - número de caracteres descartados
 *
```

```
 * Retorno: Nada
 *
 * Observação: O objetivo desta função é uniformizar as mensagens emitidas pelas
 *             funções globais da biblioteca LeituraFacil que informam que o
 *             usuário digitou mais do que deveria
 *
 ****/
static void InformaDescarte(int n)
{
      /* Verifica se há mensagem a ser apresentada */
   if (n <= 0) {
      return; /* Não há */
   }

      /* Apresenta a mensagem adequada para a situação */
   if (n == 1) {
      printf("\t>>> Um caractere foi descartado\n");
   } else {
      printf("\t>>> %d caracteres foram descartados\n", n);
   }
}
```

Análise: A implementação da função `InformaDescarte()` é bastante simples e fácil de entender. Assim, não há necessidade de comentários adicionais.

B.3.3 Funções Globais

A seguir, serão apresentadas as implementações das funções globais (i.e., públicas) da biblioteca LeituraFacil.

LeCaractere()

A função `LeCaractere()` lê um caractere via teclado e, após a leitura, remove os caracteres remanescentes no buffer. Essa função também alerta o usuário caso ele tenha digitado caracteres além do esperado.

```
/****
 *
 * LeCaractere(): Lê um caractere via teclado e remove os
 *                caracteres remanescentes no buffer
 *
 * Parâmetros: Nenhum
 *
 * Retorno: O caractere lido
 *
 * Observação: Esta função deixa o buffer limpo
 *
 ****/
int LeCaractere(void)
{
   int c, /* Armazenará cada caractere lido */
       nResto = 0; /* Número de caracteres excedentes */

   /* Volta para cá se ocorrer erro de leitura */
inicio:
   c = getchar(); /* Lê o caractere digitado */

      /* Verifica se getchar() retornou EOF */
   if (c == EOF) {
      MostraMensagem( "Erro de leitura. Tente novamente\n\t> " );
      goto inicio; /* Não faz mal algum */
```

```
   }

      /* Pelo menos '\n' se encontra no buffer */
   nResto = LimpaBuffer();

      /* Repreende o usuário se ele digitou demais */
   InformaDescarte(nResto);

   return c; /* Retorna o caractere lido */
}
```

Análise: A função `LeCaractere()` lê um caractere em **stdin** usando **getchar()**, limpa o buffer associado a esse meio de entrada por meio de uma chamada de `LimpaBuffer()` e, finalmente, informa se algum caractere foi descartado (porque o usuário digitou caracteres além do esperado).

LeInteiro()

A função `LeInteiro()` lê um valor do tipo **int** em base decimal via teclado e, após a leitura, deixa o buffer associado a esse meio de entrada limpo.

```
/****
 *
 * LeInteiro(): Lê um número inteiro via teclado e deixa o buffer intacto
 *
 * Parâmetros: Nenhum
 *
 * Retorno: O número inteiro lido
 *
 ****/
int LeInteiro(void)
{
   int num, /* O número lido */
       teste, /* Valor retornado por scanf() */
       nResto = 0; /* Número de caracteres excedentes */

   /* Desvia para cá se o valor for inválido */
inicio:
      /* Tenta ler um valor válido */
   teste = scanf("%d", &num);

      /* Se não ocorreu erro de leitura ou de final de arquivo, há */
      /* caracteres remanescentes que precisam ser removidos       */
   if (teste != EOF) {
      nResto = LimpaBuffer();
   }

      /* Enquanto o valor retornado por scanf() não indicar que um */
      /* valor válido foi lido continua tentando obter esse valor  */
   while(teste != 1) {
      MostraMensagem( "O valor digitado e' invalido. Tente novamente\n\t> " );
      goto inicio; /* Não causa dano algum */
   }

      /* Se for o caso, repreende o usuário por */
      /* ele ter digitado mais do que deveria   */
   InformaDescarte(nResto);

      /* O valor retornado certamente é válido */
   return num;
}
```

Análise: Se você entendeu bem o material apresentado na Seção 11.9, não terá dificuldade para entender a implementação da função `LeInteiro()`.

LeReal()

A função `LeReal()` lê um valor do tipo **double** via teclado deixando o buffer associado ao teclado zerado.

```
/****
 * LeReal(): Lê um valor do tipo double via teclado deixando o buffer
 *           associado ao teclado sem caracteres remanescentes
 *
 * Parâmetros: Nenhum
 *
 * Retorno: O valor lido e validado
 ****/
double LeReal(void)
{
   int    teste, /* Armazena o retorno de scanf() */
          nResto = 0; /* Número de caracteres excedentes */
   double valorLido; /* O valor digitado pelo usuário */

      /* O laço encerra quando o usuário se comporta bem */
   while (1) {
         /* Tenta ler um valor do tipo double */
      teste = scanf("%lf", &valorLido);

         /* Se não ocorreu erro de leitura ou de final de arquivo, há */
         /* caracteres remanescentes que precisam ser removidos       */
      if (teste != EOF) {
         nResto = LimpaBuffer();
      }

         /* Se o valor retornado por scanf() foi 1, a leitura foi OK */
      if (teste == 1) {
         break; /* Leitura foi nota 10 */
      } else { /* Usuário foi mal comportado */
         MostraMensagem( "O valor digitado e' invalido. Tente novamente\n\t> " );
      }
   }

      /* Se for o caso, repreende o usuário */
   InformaDescarte(nResto);

      /* Retorna um valor válido do tipo double */
   return valorLido;
}
```

Análise: Apesar das aparências, a função `LeReal()` é semelhante a `LeInteiro()`, mas a implementação de `LeReal()` não usa **goto** para agradar os rigoristas.

LeString()

A função `LeString()` lê um string via teclado deixando o buffer associado a esse meio de entrada intacto novamente após a leitura.

```
/****
 * LeString(): Lê um string via teclado deixando o buffer zerado
 *
 * Parâmetros:
 *      ar[] (saída): array que armazenará o string lido
```

```
 *      nElementos (entrada): número máximo de elementos que
 *                            serão armazenado no array, incluindo '\0'
 *
 * Retorno: O número de caracteres que o usuário digitou a mais, sem incluir '\n'
 *
 * Observações:
 *      1. "Ler um string" significa ler caracteres,
 *         armazená-los num array e acrescentar o
 *         caractere '\0' ao final para que o array contenha um string.
 *      2. O número máximo de caracteres lidos será nElementos - 1, porque se
 *         deve deixar um espaço sobressalente para o caractere '\0'.
 *      3. Se o caractere '\n' for lido, ele não fará parte do string resultante
 ****/
int LeString(char *ar, int nElementos)
{
   char *p; /* Usado para remover '\n' do string */
   int   nCarDeixados = 0; /* Conta o número de caracteres excedentes no buffer */

      /* Lê no máximo nElementos - 1 caracteres ou até encontrar '\n'.    */
      /* Quando fgets() retorna NULL, ocorreu erro ou tentativa de        */
      /* leitura além do final do arquivo. Nesses casos, o laço continua. */
   while ( fgets(ar, nElementos, stdin) == NULL )  {
      printf( "\a\n\t>>> %s", "Erro de leitura. "Tente novamente\n\t> " );
   }

      /* Faz p apontar para o caractere que antecede o caractere terminal do string */
   p = strchr(ar, '\0') - 1;

      /* Se o caractere '\n' foi lido, remove-o do   */
      /* string. Caso contrário, remove-o do buffer. */
   if (*p == '\n') {
         /* Caractere '\n' faz parte do string */
      *p = '\0'; /* Sobrescreve '\n' com '\0' */
   } else { /* Usuário digitou caracteres demais */
      nCarDeixados = LimpaBuffer();
   }

      /* Retorna o número de caracteres que o usuário */
      /* digitou além do esperado, sem incluir '\n'   */
   return nCarDeixados;
}
```

Análise: Entender a implementação da função `LeString()` é relativamente fácil desde que você tenha estudado a Seção 11.9 e seguido os comentários que acompanham essa função.

LeOpcao()

A função `LeOpcao()` lê via teclado um caractere que faz parte de um string recebido como parâmetro.

```
/****
 * LeOpcao(): Lê, via teclado, um caractere que deve fazer
 *            parte de um string para ser considerado válido
 *
 * Parâmetros:
 *      opcoes (entrada) - string contendo os caracteres válidos
 *
 * Retorno: Um caractere válido
 *
 * Observações:
```

```
 *      1. Esta função deixa o buffer virgem novamente
 *      2. Esta função é denominada 'LeOpcao' porque o caractere lido e validado
 *         é restrito a um conjunto de caracteres (opções) representado pelo
 *         string recebido como parâmetro
 ****/
int LeOpcao(const char *opcoes)
{
   int op; /* Opção (caractere) escolhida pelo usuário */

      /* Enquanto o usuário não escolher uma opção */
      /* válida, o laço a seguir não encerra       */
   while (1) {
         /* Lê o caractere digitado */
      op = LeCaractere();

         /* Verifica se o caractere é válido */
      if (strchr(opcoes, op)) {
         break; /* É */
      } else { /* Não é */
         MostraMensagem( "Opcao incorreta. Tente novamente\n\t> " );
      }
   }

      /* Certamente, a opção retornada é válida */
   return op;
}
```

Análise: A função `LeOpcao()` lê um caractere usando `LeCaractere()`. Então, ela usa a função **strchr()** (v. **Seção 8.5.8**) para verificar se o caractere lido faz parte do string recebido como parâmetro. A função `LeOpcao()` é denominada *LeOpcao* porque, tipicamente, ela é usada para ler opções disponíveis em menus (v. **Seção 5.11**).

B.4 Compilação

Observação Importante: Você não precisa compilar a biblioteca LeituraFacil para as plataformas suportadas neste apêndice (Windows/DOS, Linux e Mac OS X), pois essa biblioteca pode ser facilmente encontrada no site do livro (***www.ulysseso.com/ip***) pronta para uso. O objetivo desta seção é apenas mostrar como a biblioteca LeituraFacil pode ser compilada e preparada para uso nos ambientes de desenvolvimento suportados por este livro.

É bem mais fácil compilar uma biblioteca e deixá-la no formato correto para uso por meio de linha de comando do que usando um IDE. O objetivo aqui é obter um arquivo denominado **`libleitura.a`**, que corresponde à biblioteca LeituraFacil pronta para uso.

Existem duas opções equivalentes para obtenção do arquivo mencionado. Essas opções serão descritas adiante. Mas, antes, é importante salientar que, nos dois casos, você precisará ter uma janela de console (terminal) aberta no diretório em que se encontram os arquivos **`leitura.c`** e **`leitura.h`**, que são os arquivos-fonte da biblioteca LeituraFacil.

A primeira opção consiste em invocar um compilador para criar um arquivo-objeto a partir do arquivo-fonte **`leitura.c`**. Esse compilador deverá ser o mesmo que você usará para criar seus programas. Também, deverá haver emparelhamento entre os sistemas operacionais em questão. Quer dizer, por exemplo, você não poderá compilar a biblioteca usando GCC para Linux e usar o arquivo-objeto resultante para desenvolvimento com GCC para MinGW (Windows).

Após compilar o arquivo **leitura.c**, deve-se invocar o comando **ar** para configurar o arquivo-objeto resultante (**leitura.o**) no formato esperado pelos linkers que construirão os programas que usam a biblioteca LeituraFacil[1].

A segunda opção é bem mais simples e consiste em usar o comando **make** e um arquivo *makefile* disponível no site dedicado ao livro (*www.ulysseso.com/ip*). **Make** é um programa utilitário originário de sistemas operacionais da família Unix e distribuído junto com alguns ambientes de desenvolvimento. O utilitário **make** é particularmente útil quando é utilizado em compilação e ligação de programas grandes (i.e., consistindo de muitos arquivos) ou quando os comandos de compilação e ligação envolvem muitas opções. Em ambos os casos, é penoso digitar no console, várias e várias vezes, os comandos necessários para obter o resultado desejado.

Makefile é um arquivo escrito numa linguagem própria que o programa **make** entende. Quando o utilitário **make** é executado sem informação sobre qual arquivo processar, ele procura um arquivo denominado **makefile** (sem extensão) no diretório corrente. No site dedicado ao livro, há um arquivo *makefile* (denominado exatamente assim) para cada ambiente de desenvolvimento suportado.

Ao usar essa segunda opção, certifique-se que os arquivos **leitura.c**, **leitura.h** e **makefile** correspondentes a um mesmo ambiente de desenvolvimento residem no mesmo diretório no qual o utilitário **make** será invocado.

B.4.1 GCC (MinGW) – Windows

Se você escolher a primeira opção descrita no início desta seção, invoque o compilador GCC por meio do comando:

```
gcc -c leitura.c -o leitura.o
```

A opção de compilação **-Wall** preconizada no Capítulo 1 não precisa ser aplicada aqui porque a biblioteca LeituraFacil já foi testada inúmeras vezes e não há mensagem de advertência que possa eventualmente ser emitida. Além disso, ela não possui nenhuma construção específica do padrão C99, de modo que a opção **-std=c99** também não é necessária.

Em seguida invoque o programa **ar** como:

```
ar -r -s libleitura.a leitura.o
```

Se você decidir usar a segunda opção descrita no início desta seção, saiba que, no pacote MinGW, o nome do utilitário *make* é **mingw32-make**.

No site dedicado ao livro, há um arquivo denominado **makefile** dirigido para MinGW. Portanto, certifique-se que esse arquivo encontra-se no mesmo diretório contendo os arquivos **leitura.c** e **leitura.h** e digite o seguinte comando:

```
mingw32-make
```

Se todos os arquivos citados estiverem em seus devidos lugares, você obterá como resposta:

```
gcc -c leitura.c
ar rs libleitura.a leitura.o
ar: creating libleitura.a
```

Por outro lado, se você obtiver como resposta:

```
mingw32-make: 'libleitura.a' is up to date.
```

o significado dessa mensagem é que o arquivo **libleitura.a** já havia sido criado e não ocorreu nenhuma

[1] O comando **ar** é oriundo do sistema Unix e é responsável por criar arquivos de biblioteca no formato esperado por alguns linkers.

alteração nos arquivos-fonte que justificasse a reconstrução daquele arquivo.

B.4.2 GCC – Linux

Use os mesmos comandos vistos na **Seção B.4.1** para invocar o compilador GCC e o utilitário **ar**. Alternativamente, use o utilitário **make** do sistema Linux e o arquivo **makefile** específico para esse sistema que se encontra no site do livro. Isto é, nesse último caso, digite apenas:

```
make
```

na linha de comando de uma janela de terminal.

B.4.3 Clang – Mac OS X

As ferramentas de desenvolvimento **Clang**, que fazem parte do pacote de desenvolvimento disponível no site da Apple, funcionam de modo semelhante àquelas que fazem parte do pacote GCC (que, a propósito, também fazem parte do mesmo pacote de desenvolvimento da Apple). Ou seja, as opções de compilação e ligação comuns que funcionam com GCC também funcionam do mesmo modo com Clang. Entretanto, Clang constitui um conjunto de ferramentas bem mais moderno, sofisticado e profissional do que GCC. Para um aprendiz de programação, o que Clang oferece de melhor são as mensagens de erro e advertência, que são bem mais fáceis de ser interpretadas do que aquelas emitidas pelo compilador GCC.

Este livro usa GCC porque é o conjunto de ferramentas de desenvolvimento da linguagem C mais popular da atualidade. Enfim, se você deseja usar esse mesmo conjunto de ferramentas no sistema Mac OS X para que possa comparar os resultados que você obtém com aqueles apresentados neste livro, então, troque as referências a *clang* por *gcc* nos procedimentos descritos a seguir.

Se você escolher a primeira opção de compilação descrita no início desta seção, siga o seguinte procedimento:

[1] Copie os arquivos **leitura.c** e **leitura.h**, que se encontram no site do livro, para um diretório de trabalho.

[2] Abra uma janela de terminal.

[3] Navegue até o diretório contendo os arquivos **leitura.c** e **leitura.h**.

[4] Invoque o compilador Clang para criar um arquivo-objeto por meio do comando[2]:

```
clang -c leitura.c -o leitura.o
```

[5] Emita o comando:

```
ar -r -s libleitura.a leitura.o
```

O procedimento alternativo e mais simples consiste no seguinte:

[1] Copie os arquivos **leitura.c**, **leitura.h** e **makefile**, que se encontram no site do livro, para um diretório de trabalho.

[2] Abra uma janela de terminal.

[3] Navegue até o diretório contendo os arquivos mencionados.

[4] Invoque o utilitário **make** digitando apenas[3]:

```
make
```

[2] Se desejar usar GCC, substitua *clang* por *gcc*.

[3] Se desejar usar GCC, antes de invocar o comando **make**, abra o arquivo **makefile** com um editor de texto comum e substitua *clang* por *gcc*.

B.5 Integração

Observação Importante: Ao contrário da **Seção B.4**, esta seção também pode interessar ao aprendiz comum. Quer dizer, no **Capítulo 1**, foi mostrado como integrar (ou instalar) a biblioteca LeituraFacil com CodeBlocks e MinGW em sistemas da família Windows. Portanto, usuários desses ambientes de desenvolvimento precisam estudar esta seção apenas se desejarem aprender um pouco mais sobre integração da referida biblioteca. Por outro lado, é exatamente nesta seção que usuários de Linux e Unix (em particular Mac OS X) aprenderão como integrar essa biblioteca.

Integrar a biblioteca LeituraFacil a um ambiente de desenvolvimento significa fazer com que o compilador e o linker utilizados encontrem os arquivos necessários dessa biblioteca nos formatos corretos. Precisamente, o arquivo que interessa ao linker é o arquivo-objeto no formato requerido, que a **Seção B.4** mostrou como pode ser obtido. Por outro lado, o arquivo que interessa ao compilador é o arquivo de cabeçalho `leitura.h`, que deve ser copiado para o diretório no qual o compilador costuma procurar arquivos de cabeçalho. Na verdade, arquivos de cabeçalho podem ser colocados em qualquer diretório. Mas, se você não armazenar um arquivo de cabeçalho num local que o compilador não considera padrão, precisará especificar o caminho até o diretório onde ele se encontra.

Os diretórios nos quais os referidos arquivos devem residir dependem das ferramentas de desenvolvimento bem como do sistema operacional utilizados. Assim, se as subseções a seguir não contemplarem seu ambiente de desenvolvimento, consulte a documentação que o acompanha para descobrir onde e como esses arquivos devem ser armazenados.

Importante: O processo de integração da biblioteca LeituraFacil também envolve informar o linker que ele deve usar o arquivo-objeto que contém a biblioteca compilada. Em qualquer ambiente de desenvolvimento suportado neste apêndice, a opção que deve ser incluída na invocação do linker é:

```
-lleitura
```

B.5.1 GCC (MinGW) – Windows

O arquivo de cabeçalho `leitura.h` deve residir no diretório `...\MinGW32\include`, enquanto que o arquivo-objeto `libleitura.a` deve ser armazenado no diretório `...\MinGW32\lib`, sendo que `...\` representa o caminho até o diretório onde jaz a instalação do pacote MinGW. Portanto, para copiar os dois arquivos mencionados para os diretórios adequados, encontre o diretório de instalação de MinGW e copie e cole os referidos arquivos para seus respectivos diretórios. (**Sugestão**: Use a ferramenta de busca do Windows para descobrir onde se encontram os referidos diretórios.)

B.5.2 GCC – Linux

O modo mais fácil de integrar a biblioteca LeituraFacil para uso com GCC no Linux é executando o arquivo de script `copialib`, que se encontra no site dedicado ao livro. Para obter o resultado desejado, siga o seguinte procedimento:

[1] Abra uma janela de terminal.

[2] Navegue até o diretório contendo os arquivos `leitura.h`, `libleitura.a` e `copialib`.

[3] Na linha de comando, digite[4]:

```
sudo ./copialib
```

[4] Quando instado pelo Linux, digite sua senha de super-usuário.

[4] Se este comando não funcionar em sua distribuição de Linux, tente o seguinte comando alternativo: `su -c ./copialib`.

Como existe um número incontável de distribuições de Linux, não há garantia que o script mencionado acima funcionará com sua distribuição particular de Linux. Neste caso, você deve ser um legítimo usuário dos bons tempos de Linux para saber executar os passos descritos a seguir.

[1] Abra uma janela de terminal (X Windows).

[2] Navegue até o diretório contendo os arquivos **`leitura.h`** e **`libleitura.a`**.

[3] Na linha de comando, digite os seguintes comandos:

```
cp leitura.h /usr/local/include/leitura.h
cp libleitura.a /usr/local/lib/libleitura.a
```

Esses comandos devem copiar os arquivos **`leitura.h`** e **`libleitura.a`**, respectivamente para os diretórios: **`/usr/local/include`** e **`/usr/local/lib`**.

[4] Após copiar os arquivos para seus devidos diretórios, você deverá invocar o utilitário **`ranlib`** para deixar o arquivo **`libleitura.a`** pronto para uso da seguinte maneira:

```
ranlib /usr/local/lib/libleitura.a
```

Se você não for capaz de efetuar o procedimento acima, troque sua distribuição de Linux ou deixe de usar Linux definitivamente. Talvez essa não seja sua praia...

B.5.3 Clang – Mac OS X

A maneira mais simples de integrar a biblioteca LeituraFacil para uso com Clang ou GCC é executando o arquivo de script **`copialib.sh`** que se encontra no site do livro. Para obter o resultado desejado, siga o seguinte procedimento:

[1] Abra uma janela de terminal.

[2] Navegue até o diretório contendo os arquivos **`leitura.h`**, **`libleitura.a`** e **`copialib.sh`**.

[3] Na linha de comando, digite:

```
sudo ./copialib.sh
```

[4] Quando instado pelo sistema, digite sua senha de super-usuário.

Diferentemente dos pobres usuários de Linux, os nobres usuários de Mac OS X têm a sua disposição um legítimo sistema Unix puro-sangue. Portanto, dificilmente, o procedimento acima não funcionará.

Mas, caso você tenha alterado sua interface de comandos (*shell*) de modo que o script citado deixe de funcionar, siga o seguinte procedimento:

[1] Abra uma janela de terminal.

[2] Navegue até o diretório contendo os arquivos **`leitura.h`** e **`libleitura.a`**.

[3] Na linha de comando, digite os seguintes comandos (nesta ordem):

```
cp leitura.h /usr/include/leitura.h
chmod a+r /usr/include/leitura.h
cp libleitura.a /usr/lib/libleitura.a
chmod a+x /usr/lib/libleitura.a
ranlib /usr/lib/libleitura.a
```

[4] Se solicitado pelo sistema, digite sua senha de super-usuário.

Se nenhum dos procedimentos enumerados acima não funcionar, certamente você pressionou o botão errado e detonou sua instalação do sistema.