PREFÁCIO

AO LEITOR

Você tem em mãos o segundo e último volume de *Programando em C*. Este volume é dedicado principalmente à biblioteca padrão de C, examinando minuciosamente todos os componentes (tipos, macros, funções e variáveis globais) dos 24 cabeçalhos que a integram. Além disso, outros tópicos importantes em programação, como localização de programas, portabilidade de programas, processamento de caracteres extensos e multibytes, e tratamento de sinais e exceção, também são explorados. Acompanhando a discussão destes tópicos, são apresentados componentes da biblioteca padrão de C que podem ser usados na solução de problemas comuns associados a cada tópico. Por exemplo, o **Capítulo 5** faz uma introdução à localização de programa e, ao mesmo tempo, apresenta, com exemplos, os componentes do cabeçalho `<locale.h>`, que oferece suporte para resolução de problemas associados à localização de programas.

O presente volume possui as seguintes características:

- Explora completamente todos os componentes da biblioteca padrão de C, levando em consideração o padrão ISO/IEC 9899:1999 (C99), a Correção Técnica 1 (2001) e a Correção Técnica 2 (2004).
- Apresenta cerca de 400 exemplos de programação. Todos estes exemplos foram testados pelo autor e por monitores sob sua supervisão em compiladores que aderem ao padrão C99.
- Contém exercícios propostos de revisão do assunto ao final de cada capítulo.
- Contém um capítulo específico sobre portabilidade de programas.
- Discute e apresenta exemplos práticos de uso de padrões para códigos de caracteres universais (ISO 10464 e Unicode) e esquemas de representação UTF.
- Aborda tópicos intimamente associados a internacionalização de programas, tais como localização de programas, caracteres multibytes e colação de *strings*.
- Apresenta diversos exemplos de programas que usam localidades nacionais nos sistemas operacionais Linux e Windows.

- Apresenta uma referência completa de especificadores de formato para as famílias scanf e printf e para datas e horas.
- Apresenta uma lista de erros comuns de programação em C que ajuda o programador a encontrar ou prevenir eventuais erros em seus programas.

Público-alvo

Este volume foi elaborado com dois objetivos gerais: (1) servir como referência a estudantes de programação e programadores que usam a linguagem C e (2) ensinar alguns tópicos avançados de programação usando esta linguagem. O livro destina-se primariamente a professores e estudantes de cursos da área de Computação e Informática, mas dificilmente poderá ser usado isoladamente como livro-texto. Além disso, a maior parte do texto não constitui material básico e requer algum conhecimento intermediário de programação em C.

O livro é recomendado idealmente como curso intermediário de programação quando o aluno já tem conhecimento básico de programação em C. Portanto, se este volume for utilizado como um curso de programação, ele deve ser complementado com um texto que preencha estes pré-requisitos. Em geral, o conteúdo do **Volume I** é suficiente como pré-requisito.

Escopo

Alguns componentes da biblioteca padrão de C requerem conhecimento específico para ser utilizados. Em casos onde o conhecimento necessário está na área de programação, como tratamento de sinais e processamento de caracteres multibytes, é feita uma introdução ao assunto. Mas, o mesmo não ocorre quando o conhecimento requerido pertence a outra área de conhecimento, como números complexos ou estatística.

O padrão ISO C99 define dois tipos de sistemas: (1) sistemas com hospedeiro (sistema operacional) e (2) sistemas livres. Assim como o **Volume I**, o presente volume dá ênfase a sistemas com hospedeiro, mas o leitor não deverá ter dificuldades para utilizar o conhecimento apresentado aqui em sistemas livres, visto que, nestes sistemas, o padrão C99 requer apenas suporte limitado à biblioteca padrão de C (v. **Capítulo 1**).

Organização

Uma sinopse do conteúdo do **Volume I** será apresentada a seguir. Logo depois, o conteúdo de cada capítulo deste volume será descrito.

Volume I

O **Capítulo 1** apresenta as construções básicas da linguagem C, como seus tipos de dados elementares e suas estruturas de controle.

O **Capítulo 2** ensina como construir programas monoarquivos usando um ambiente IDE ou um editor de programas em conjunto com um compilador. Este capítulo também ensina como usar as funções mais elementares de entrada e saída.

O **Capítulo 3** explora definição e uso de funções, assim como os conceitos fundamentais de endereços e ponteiros. Funções recursivas, *inline* e com listas de argumentos variáveis também são apresentadas neste capítulo.

O **Capítulo 4** ensina a construir programas multiarquivos e expõe conceitos tais como classes de armazenamento, tipos de dados derivados e qualificadores de tipos.

O **Capítulo 5** introduz o pré-processador de C e suas construções próprias.

O **Capítulo 6** discute legibilidade e depuração de programas.

O **Capítulo 7** enfoca arrays e ponteiros, e as relações entres eles.

O **Capítulo 8** apresenta *strings* e as funções para processamento de *strings* mais comuns. Este capítulo mostra ainda como definir a função **main()** com parâmetros.

Estruturas, uniões e enumerações são apresentadas no **Capítulo 9**, que inclui ainda iniciadores designados, arrays flexíveis e literais compostos.

O **Capítulo 10** discute as seguintes construções: arrays de ponteiros, ponteiros para ponteiros e ponteiros para funções.

O **Capítulo 11** introduz alocação dinâmica de memória, incluindo listas encadeadas e suas operações fundamentais.

O **Capítulo 12** expõe entrada e saída de dados (processamento de arquivos).

O **Capítulo 13** faz uma introdução à programação de baixo nível em C e apresenta aplicações práticas, como o uso de sinalizadores e criptografia.

O **Apêndice A** contém uma tabela completa com precedências e associatividades de todos os operadores da linguagem C, e o **Apêndice B** apresenta, de modo resumido, os especificadores de formato básicos utilizados por funções das famílias printf e scanf.

VOLUME II

O **Capítulo 1** apresenta uma introdução e uma visão geral da biblioteca padrão de C. Aqui, além da apresentação do propósito de cada cabeçalho, os componentes são descritos de maneira suficientemente breve para permitir que o leitor tenha uma ideia do papel de cada componente. Assim, a síntese da biblioteca padrão apresentada neste capítulo tem mais utilidade como referência ou quando o programador já possui conhecimento sobre um determinado componente ou cabeçalho e deseja apenas refrescar a memória. Porém, o mais importante são as recomendações gerais e os alertas quanto ao uso dos componentes da biblioteca padrão que o programador deve levar em consideração.

O **Capítulo 2** começa com uma breve introdução aos tipos inteiros da linguagem C cujo objetivo é prover o conhecimento mínimo necessário para o entendimento dos cabeçalhos apresentados neste capítulo: `<limits.h>`, `<stdint.h>` e `<inttypes.h>`. Além de discutir os componentes destes cabeçalhos, este capítulo também apresenta funções declaradas no cabeçalho `<stdlib.h>` que executam operações aritméticas sobre inteiros.

O **Capítulo 3** introduz brevemente os tipos de ponto flutuante reais e explora em profundidade os componentes dos cabeçalhos `<float.h>`, `<math.h>` e `<fenv.h>`. É interessante notar que algumas funções do cabeçalho `<math.h>` são utilizadas em áreas específicas, tais como Física Matemática [e.g., **tgamma()**] e Estatística [e.g., **erf()**]. Portanto, entender exatamente aquilo que estas funções calculam requer um conhecimento mais profundo em Matemática do que o que é tipicamente ensinado em cursos de Computação. Prover este tipo de conhecimento está bem além do escopo deste livro.

O **Capítulo 4** faz uma breve introdução aos tipos de ponto flutuante complexos e apresenta os componentes dos cabeçalhos `<complex.h>` e `<tgmath.h>`. O entendimento da descrição da maioria das funções do cabeçalho `<complex.h>` requer conhecimento prévio de cálculo com variáveis complexas, que é tipicamente ensinado em cursos de Engenharia Elétrica/Eletrônica, mas o mesmo não ocorre em cursos de Computação. Se você não possui este conhecimento prévio, é recomendável consultar um bom texto sobre variáveis complexas antes de estudar este capítulo.

Nestes tempos de globalização, o **Capítulo 5** é extremamente importante, pois apresenta um tópico intimamente relacionado ao tema: localização de programas. Este capítulo discute os cabeçalhos <locale.h> e <time.h>, cujos componentes proveem suporte para localização e datação. Aspectos elementares de localização, medições de tempo e formatação de datas e horas também são discutidos neste capítulo.

O **Capítulo 6** lida com caracteres monobytes (i.e., do tipo **char**) e *strings* de caracteres monobytes. Estes tipos de caracteres e *strings* correspondem àqueles discutidos no **Volume I**. Neste capítulo, são estudados os cabeçalhos <ctype.h> e <string.h> e as funções declaradas no cabeçalho <stdlib.h> dedicadas à conversão de *strings* em números.

No **Capítulo 7**, faz-se uma introdução a dois temas complexos e importantes: caracteres extensos e multibytes. Devido à relativa complexidade dos temas expostos, este capítulo é essencialmente conceitual e deve ser visto como uma preparação básica para as aplicações práticas apresentadas no capítulo a seguir. Este capítulo também descreve o algoritmo de colação Unicode, que representa a abordagem de tratamento de colação mais aceita atualmente.

O **Capítulo 8** corresponde à aplicação prática dos conceitos apresentados no **Capítulo 7**. Este capítulo versa sobre os cabeçalhos <wctype.h> e <wchar.h>, mas as funções de entrada e saída envolvendo caracteres extensos declaradas neste último cabeçalho são apresentadas no **Capítulo 10**. Por outro lado, as funções usadas em conversão entre caracteres extensos e multibytes declaradas no cabeçalho <stdlib.h> são apresentadas aqui.

Funções com lista de argumentos variáveis são apresentadas no **Capítulo 9**. Apesar de o tema já ter sido abordado com relativa suficiência no **Volume I**, ele volta à tona neste capítulo. Aqui, além da macro **va_copy()**, que não foi apresentada no **Volume I**, são expostos maiores detalhes sobre essa categoria de funções e vários exemplos novos são discutidos.

O **Capítulo 10** contém basicamente os mesmos tópicos do **Capítulo 12** do **Volume I**, mas a ênfase e o enfoque de exposição são diferentes nos dois casos. Aqui, os conceitos são apresentados com mais brevidade e as descrições dos componentes do cabeçalho <stdio.h> são mais detalhadas. Além disso, este capítulo aborda entrada e saída envolvendo caracteres e strings extensos e multibytes, o que não ocorre no **Volume I**.

O **Capítulo 11** apresenta os cabeçalhos: <stdbool.h>, <iso646.h>, <errno.h>, <signal.h> e <setjmp.h>. De algum modo, o uso dos dois primeiros cabeçalhos pode melhorar a legibilidade e a portabilidade para outras linguagens (e.g.,

C++) de programas escritos em C. Os três últimos cabeçalhos apresentados neste capítulo são tipicamente usados em tratamento de erros e exceções. Este capítulo também apresenta conceitos e exemplos relacionados a estes dois tópicos.

O **Capítulo 12** encerra o estudo da biblioteca padrão de C com a apresentação de dois cabeçalhos de propósito geral: `<stdlib.h>` e `<stddef.h>`. Parte dos componentes do cabeçalho `<stdlib.h>` é distribuída em outros capítulos devido à maior proximidade contextual: as funções declaradas neste cabeçalho dedicadas a operações aritméticas com inteiros são apresentadas no **Capítulo 2**, aquelas que realizam conversões de *strings* em número são apresentadas no **Capítulo 6**, e as funções envolvidas com conversões entre caracteres multibytes e extensos são exploradas no **Capítulo 8**.

O **Capítulo 13** discute portabilidade de programas escritos em C e este tópico não deve ser negligenciado, visto que esta é uma das principais deficiências da linguagem C.

O **Apêndice A** apresenta um resumo de todos os elementos de composição de programas escritos em C, que são, coletivamente, denominados *construtores*. Este apêndice menciona a seção desta obra onde cada construtor é explorado em detalhes, tornando-se, assim, bastante útil como referência. O **Apêndice B** apresenta exaustivamente todos os possíveis especificadores de formato que podem compor um *string* de formatação para qualquer membro das famílias printf e scanf. O **Apêndice C** descreve os possíveis componentes de *strings* de formatação de datas e horas que podem ser usados com as funções **strftime()** e **wcsftime()**. Finalmente, o **Apêndice D** apresenta uma lista de erros comuns de programação em C, que pode ser usada como lista de verificação, preventiva ou corretiva, de programas.

Ao final de cada capítulo, são incluídos **Exercícios de Revisão** que objetivam a verificação de aprendizagem do material exposto no respectivo capítulo. As respostas destes exercícios podem ser encontradas diretamente no texto ou usando-se um mínimo de dedução ou experimentação.

Exemplos e códigos-fonte

A maioria dos exemplos de programas foi testada usando o compilador gcc acompanhado de `-std=c99` como opção de compilação e `-lm` como opção de ligação[1]. A tabela a seguir descreve os ambientes nos quais a maioria dos programas apresentados como exemplos neste livro foi testada. Quando houver alguma divergência entre os dados apresentados nesta tabela e o modo como um dado programa foi compilado ou executado, o leitor será devidamente informado. Quando não houver menção a compilador ou a sistema operacional, espera-se que o resultado produzido seja o mesmo independentemente destes parâmetros.

Quando o texto informa que um programa foi...	Significa que...
Compilado e executado no Linux	• O programa foi compilado com gcc 4.3.2 usando a biblioteca libc6 2.8. • O sistema no qual ele foi compilado e executado foi Linux Ubuntu 8.10. • O computador utilizado possuía CPU Pentium D945 com 2 GB de memória RAM.
Compilado e executado no Windows	• O programa foi compilado no ambiente Dev-C++ 4.9.9.2 usando glibc 2.2.3. •O sistema no qual ele foi compilado e executado foi Windows XP SP2. • A máquina utilizada foi um computador com CPU Pentium Quad Q6600 com 4 GB de memória RAM.

Os resultados de execuções de alguns programas utilizados como exemplos são apresentados quando se julga que eles contribuem para melhorar o entendimento. Por outro lado, quando um resultado é trivial, ele não é apresentado. De qualquer modo, como todos os programas estarão disponíveis na internet (v. adiante), os resultados poderão ser verificados compilando-se e executando-se estes programas.

Exemplos de programas que envolvem localidade foram, em sua maioria, testados apenas no sistema operacional Linux (v. tabela anterior). A única alteração necessária para estes programas funcionarem em outro sistema operacional diz respeito à especificação de localidades no respectivo sistema. Em outros sistemas operacionais da família Unix, talvez nenhuma alteração desses programas seja necessária. Por outro

1 Esta última opção só afeta programas que usam funções declaradas em `<math.h>`, `<complex.h>` e `<tgmath.h>`.

lado, a especificação de localidades em sistemas operacionais da família Windows é mais complicada e depende da versão do sistema considerada.

No site do livro da internet, *http://www.ulysseso.com/progc2.htm*, encontram-se os códigos-fonte de todos os exemplos apresentados no livro, além de outros programas não inseridos no texto. Este material é classificado de acordo com os capítulos correspondentes no livro e encontra-se comprimido em formato *zip*. Para baixá-lo, pode-se usar qualquer navegador de *web*.

Recomendações ao estudante

Técnicas de programação não podem ser dominadas simplesmente estudando-se material escrito. Algumas sugestões para um melhor aprendizado das técnicas de programação apresentadas aqui são:

- *Edite, compile e execute os programas apresentados como exemplos no texto, tentando entender como eles funcionam*. Arquivos contendo estes exemplos podem ser obtidos via internet (v. mais adiante).

- *Utilize os exemplos apresentados como base para experimentos com os componentes da biblioteca padrão*. Muitos componentes, notadamente funções, podem ser usados de diversas maneiras que não foram totalmente exploradas nos exemplos apresentados. Nestes casos, recomenda-se que o estudante altere tais exemplos para testar outras opções oferecidas pelos respectivos componentes a que os exemplos se referem.

- *Use um depurador como ferramenta de aprendizagem de programação*. Utilizando um depurador para executar um programa mesmo quando ele não precisa ser depurado pode-se aprender muito mais sobre seu funcionamento. Esta sugestão é detalhada no **Volume I** (**Capítulo 6**).

- *Desenvolva um estilo de programação*. Não existe um estilo único de escrita de programas que seja considerado o mais correto, mas, se você aderir aos conselhos básicos de estilo de programação sugeridos no **Volume I** e seguidos no presente volume, seu estilo estará bem fundamentado.

- *Tente responder todos os exercícios de revisão propostos ao final de cada capítulo*. Este volume não inclui seções contendo exclusivamente exercícios de programação, como ocorre no **Volume I**, mas muitos exercícios propostos no presente volume são de natureza prática e requerem a escrita de programas.

Além disso, muitas questões de natureza conceitual não podem ser relegadas a segundo plano.

O material complementar que o aluno deve ter disponível para acompanhamento do texto consiste em um computador e um ambiente de programação C. No site dedicado ao livro na internet encontram-se referências abundantes para páginas da web onde se podem encontrar ferramentas de programação, exemplos de códigos-fonte, artigos e outros materiais úteis na aprendizagem e no aprimoramento do conhecimento sobre programação em C.

Convenções adotadas no livro

Convenções tipográficas

Itálico é usado nas seguintes situações:

- Para enfatizar determinado ponto.
- Para representar componentes de construções da linguagem C ou de comandos de sistemas operacionais, compiladores, depuradores, etc. que devem ser substituídos por aquilo que realmente representam. Por exemplo, no comando do compilador gcc: `gcc -c` *`nome-de-arquivo`*, *`nome-de-arquivo`* é um guardador de lugar que deve ser substituído por um verdadeiro nome de arquivo quando o comando for utilizado.
- Em palavras que representam estrangeirismos. (Palavras de origem estrangeira, como array e buffer, reconhecidas pelos principais dicionários brasileiros não são representadas desta maneira.)

Utiliza-se **negrito** quando:

- Conceitos são definidos.
- Palavras-chave e identificadores reservados da linguagem C aparecem no corpo do texto, mas não é o caso quando eles aparecem em programas ou trechos de programas.

- Operadores da linguagem C são mencionados fora de programas ou trechos de programas.
- Em referências a capítulos, seções, tabelas, etc.

A fonte `courier` é utilizada nos seguintes casos:

- Na apresentação de programa ou trechos de programas.
- Na apresentação de comandos que aparecem numa interface de linha de comandos (*shell*).
- Em nomes de arquivos e diretórios.
- Na representação de constantes numéricas, caracteres e *strings*.
- Na representação gráfica de teclas ou combinações de teclas. Neste caso, as teclas são escritas em maiúsculas e colocadas entre colchetes, como, por exemplo, `[CTRL-Z]`.

A fonte **`courier em negrito`** é utilizada para representar dados introduzidos por um usuário em exemplos de interação entre um programa e um usuário, enquanto que a fonte *`courier em itálico`* é utilizada para representar conteúdo impresso por um programa no meio de saída padrão.

Apresentação de componentes da biblioteca padrão de C

As funções e macros com argumentos da biblioteca padrão são apresentadas seguindo o seguinte esquema (nesta ordem):

- ***Nome-da-função*** ou ***nome-da-macro***
- **Incluir:** Cabeçalho que deve ser incluído para habilitar o uso da função ou macro.
- **Descrição:** Uma breve descrição da função ou macro.
- **Protótipo**: Protótipo da função ou da macro (v. a seguir).

- **Parâmetros:** Descrições dos parâmetros da função ou da macro.
- **Retorno:** Valor retornado pela função ou resultante da invocação da macro.
- **Observação(ões):** Informações adicionais que esclarecem o uso da função ou da macro, ou referência para outras seções do livro.
- **Exemplo:** Um exemplo de uso da função ou da macro, ou uma referência para o local onde tal exemplo pode ser encontrado.

Evidentemente, macros não possuem protótipo tal como este conceito é definido para funções, mas, para efeitos práticos de uso, pode-se imaginar que macros com argumentos possuem protótipos similares àqueles usados com funções.

Tipos, macros sem argumentos e variáveis globais da biblioteca padrão são apresentados em forma de tabela, a não ser que a relativa complexidade do componente justifique uma discussão mais longa. Neste caso, a discussão segue o seguinte esquema (nesta ordem):

- ***Nome-do-componente***
- **Incluir:** Cabeçalho que deve ser incluído para permitir o uso do componente.
- **Descrição:** Descrição completa do componente.
- **Exemplo:** Um exemplo de uso do componente, ou uma referência para o local onde tal exemplo pode ser encontrado.

Para efeito de apresentação, rótulos de estruturas são tratados como se fossem tipos devido à proximidade com tipos derivados de C. De fato, se `rotulo` é um rótulo de estrutura, `struct rotulo` é um tipo.

Outras convenções

Alterações introduzidas pelo mais recente padrão ISO de C são identificadas no texto por **(C99)**. É importante que o leitor dê atenção a esta indicação, pois uma dada característica introduzida por C99 pode não ter sido ainda implementada em seu compilador. Além disso, alguns compiladores requerem uma opção explícita (e.g., `-std=C99` no compilador gcc) para ativação do padrão C99.

O uso de acentuação infelizmente continua apresentando problemas para programadores cuja língua natural requer uso intensivo de letras acentuadas. Isso ocorre porque não há editores de programas que ofereçam suporte adequado ao uso de caracteres no formato UCN sugerido pelo padrão C99. Portanto, a maioria dos programas exibidos como exemplos apresenta palavras acentuadas no meio de saída apenas quando o objetivo é exatamente demonstrar como isso pode ser realizado.

Constantes são apresentadas do mesmo modo como elas são interpretadas em C. Na representação gráfica de números binários, o subscrito 2 (por exemplo, 11010010_2) é utilizado apenas quando há iminência de ambiguidade. De modo análogo, quando existe impendente ambiguidade na representação gráfica de números octais e decimais, os subscritos 8 e 10 são utilizados, respectivamente.

Como, em C, o nome de uma função considerado isoladamente representa seu endereço, quando se faz referência a uma função (e não ao seu endereço), usa-se seu nome seguido de um par de parênteses [e.g., **printf()**]. Referências a arrays seguem raciocínio semelhante (e.g., `ar[]`), já que o nome de um array também representa seu endereço. Macros com argumentos não seguem o mesmo raciocínio, mas, devido à analogia entre macros com argumentos e funções, elas seguem a mesma notação usada com funções [e.g., **va_copy()**].

Três pontos num fragmento de programa representam um trecho de programa (i.e., declarações e instruções) omitido por não ser relevante para a discussão em foco. Mas, três pontos em alusão ou cabeçalho de função representam uma lista de argumentos variáveis. Isto é, neste caso, os três pontos fazem realmente parte do cabeçalho ou da alusão.

As convenções utilizadas na escrita de identificadores são aquelas apresentadas no **Volume I** (**Capítulo 6**) e reproduzidas aqui para facilidade de referência:

- *Nomes de variáveis* começam com letra minúscula; quando o nome da variável é composto, utiliza-se letra maiúscula no início de cada palavra seguinte, incluindo palavras de ligação. Exemplo: `notaDoAluno`.

- *Nomes de tipos* seguem as mesmas regras para nomes de variáveis, mas começam sempre com a letra *t*. Exemplo: `tLista`.

- *Nomes de funções* começam com letra maiúscula e seguem as demais regras para nomes de variáveis. Exemplo: `OrdenaLista`.

- *Nomes de macros* utilizam apenas letras maiúsculas; se um nome de macro for composto, utiliza-se sublinha para separar os componentes. Exemplo: `VALOR_ABSOLUTO`.

Cabeçalho que fazem parte da biblioteca padrão de C são colocados entre "<" e ">" e escritos em `courier`. Por exemplo: `<stdio.h>`.

SIMPLIFICAÇÕES

Compilação versus construção de programa executável. Compilar um arquivo-fonte não é o mesmo que transformá-lo em programa executável. Esta distinção é claramente apresentada no **Volume I**. Mesmo assim, como é comum, algumas vezes, *compilar um programa* é utilizado com o significado de *construir um programa executável*. Espera-se que o leitor possa deduzir do contexto o real significado de *compilação*.

Declaração versus definição. Definir uma variável ou função significa, em poucas palavras, causar a geração de código capaz de implementar a variável ou função. Por outro lado, declarar uma variável ou função significa simplesmente aludir à variável ou função. A linguagem C faz inequívoca distinção entre estes conceitos. Apesar disso, pode-se eventualmente utilizar *declaração* no texto quando o significado pretendido é *definição*. Novamente, espera-se que o leitor possa discernir os dois significados de *declaração*.

Stream versus arquivo. *Stream* é um conceito utilizado por C e outras linguagens de programação que permite que se processem arquivos sem dar atenção à origem ou destino de dados. Arquivo, por sua vez, representa qualquer dispositivo de onde se podem ler dados ou onde se podem depositá-los. Apesar de muitos leitores estarem familiarizados com o conceito mais usual de arquivo (e.g., uma coleção de bytes armazenada em disco rígido), eles não aparentam ter dificuldade em entender o conceito generalizado de arquivo utilizado por C e Unix. Assim, o termo *arquivo* é muitas vezes utilizado onde o termo mais adequado deveria ser *stream*.

Unix versus Linux. Do ponto de vista das referências feitas a estes sistemas neste livro, não há nenhuma diferença entre eles.

Cabeçalho versus arquivo de cabeçalho. O padrão ISO da linguagem C não especifica que devam existir arquivos de cabeçalhos contendo declarações dos diversos componentes da biblioteca padrão de C. O que esse padrão especifica é que devem existir *cabeçalhos* (e não exatamente *arquivos de cabeçalho*) cujos conteúdos especí-

ficos devem tornar-se disponíveis quando o programador utiliza tais cabeçalhos com diretivas **#include**. O modo como um dado compilador implementa isso não é especificado pelo padrão. Assim, uma diretiva como `#include <stdio.h>` não significa necessariamente incluir um arquivo denominado `stdio.h` no local onde esta diretiva se encontra. Em termos práticos, no entanto, esta sutileza só é importante para fabricantes de compiladores; i.e., para o programador, uma diretiva **#include** funciona do mesmo modo, quer um arquivo de cabeçalho seja realmente incluído ou não. Assim, tanto *arquivo de cabeçalho* quanto *cabeçalho* podem ser usados sem distinção.

Indireção. Esta palavra inexiste no vernáculo e é usada para representar *dereferencing* em inglês, que é o ato de acessar o conteúdo de uma porção de memória utilizando um ponteiro. Isto é, indireção significa acesso *indireto* a um conteúdo em memória por meio de um ponteiro, em vez do acesso direto promovido por variáveis comuns.

Arrays como argumentos e retorno de funções. Rigorosamente, em C, funções não recebem nem retornam arrays. Portanto, quando se fala no texto que uma função *recebe um array como argumento* ou *retorna um array*, o que se quer dizer é, respectivamente, que a função *recebe um ponteiro para o primeiro elemento de um array* ou *retorna um ponteiro para o primeiro elemento de um array*. A mesma simplificação elíptica aplica-se a *strings*, que também são arrays. Por outro lado, quando se menciona *ponteiro para um array* (ou *string*), o significado pretendido é *ponteiro para o endereço de um array* (ou *string*). Portanto, neste último caso, trata-se de um ponteiro para ponteiro.

Caractere versus caractere monobyte. No **Volume I** desta obra e na grande maioria dos livros de programação em C, existe apenas um tipo de caractere: aquele que pode ser armazenado numa variável do tipo **char** (i.e., num único byte). O presente volume lida com três categorias de caracteres: monobytes, multibytes e extensos, mas, por simplicidade, o termo *caractere monobyte* é usado apenas quando houver iminência de ambiguidade. Em outras palavras, o termo *caractere* (sem qualificação) denota *caractere monobyte*.

String monobyte, string multibyte e string extenso. Literalmente, *string monobyte* é um string constituído de um único byte. Neste livro, entretanto, para simplificar a terminologia, *string monobyte* tem o significado de *string constituído de caracteres monobytes*. De modo semelhante, *string multibyte* é um *string* constituído de caracteres multibytes, e *string extenso* é um *string* constituído de caracteres extensos.

Unicode e ISO 10646 são códigos de caracteres? Algumas vezes, Unicode e ISO 10646 são referidos no texto como códigos de caracteres quando, de fato, se deseja fazer referência aos códigos de caracteres especificados por estes padrões. Mas, con-

forme é explicitamente afirmado no texto, Unicode e ISO 10646 (principalmente, Unicode) vão bem mais além do que simplesmente estabelecer um mapeamento entre pontos de código e caracteres.

PRÁTICAS INIMITÁVEIS

O livro adota práticas que o programador deve evitar em programação real. A adoção destas práticas possui justificativas no contexto do livro que são apresentadas a seguir:

- *Comentários didáticos*. Na prática, comentários devem ser escritos para programadores que conhecem a linguagem e não têm caráter didático como muitos comentários apresentados no texto. Naturalmente, os comentários apresentados aqui são didáticos porque estão inseridos num livro didático.

- *Números mágicos*. No **Volume I**, recomenda-se que números mágicos sejam substituídos por constantes simbólicas. Mesmo assim, há alguns números mágicos em exemplos apresentados no texto. A justificativa é que, nestes exemplos, não existe contexto necessário para encontrarem-se denominações significativas para associar a esses números.

- *Verificação de valores retornados por funções*. Muitas das funções da biblioteca padrão de C retornam valores que indicam se uma dada operação foi bem-sucedida ou não. Na prática, o programador deve sempre testar estes valores, em vez de assumir que uma dada operação sempre obtenha êxito. Em alguns exemplos apresentados no texto, estes valores não são testados para não desviar atenção daquilo que o exemplo pretende enfatizar.

- *Uso de tipos inteiros primitivos*. Os únicos tipos inteiros primitivos de C portáveis são **signed char**, **unsigned char** e **long long** (C99). Devido à natureza didática e a relativa simplicidade dos exemplos exibidos no livro, existe pouca possibilidade de ocorrerem problemas de portabilidade. Por isso, tipos inteiros não portáveis são fartamente utilizados. Na prática, para evitar problemas de portabilidade, o programador deve usar os tipos inteiros de larguras fixas definidos no cabeçalho `<stdint.h>` (v. **Capítulo 2**).

Críticas, sugestões e comentários

Como qualquer texto (ou programa) de considerável dimensão, este livro pode conter erros ou imperfeições que não puderam ser detectadas e corrigidas em tempo. Portanto, qualquer crítica ou indicação de erros encontrados no livro é bem-vinda.

Se, porventura, algum programa encontrado no livro ou no site dedicado a ele apresentar um comportamento inesperado que possa caracterizar um erro de programação (*bug*), não hesite em entrar em contato.

Qualquer questão de natureza técnica ou comentário útil pode ser enviado utilizando formulário próprio no site do livro na internet: *http://www.ulysseso.com/progc2.htm*. Neste site, também se encontra vasto material que complementa o livro, em particular, e sobre programação em C, em geral.

Ulysses de Oliveira

Julho de 2009