# PREFÁCIO

## Ao Leitor

### *O Desafio de Ensinar e Aprender a Programar*

Programação requer conhecimentos que podem ser separados em três áreas que devem ser dominadas ao mesmo tempo:

[1] **Metodologia de construção de algoritmos**. Normalmente, um iniciante em programação não domina esse conhecimento. Por exemplo, ele é certamente capaz de decidir se um número é primo, mas talvez seja incapaz de descrever os passos necessários para resolver esse problema. O domínio desse conhecimento deve ser o foco de um curso introdutório em programação.

[2] **Linguagem de programação**. Poucas coisas frustram mais um aprendiz do que não obter opinião que o permita verificar o progresso de seus esforços em resolução de problemas em uma dada disciplina. Apesar de se apregoar que um algoritmo deve ser testado à mão, convenhamos que o modo mais simples e gratificante de testar um algoritmo é construindo um programa que o realize e observando os resultados. Assim, o foco de um curso de programação não deve ser uma linguagem em si, mas o uso de uma linguagem de programação para demonstrações práticas parece ser inexorável.

[3] **Ambiente de desenvolvimento**. A justificativa para a necessidade de um ambiente de desenvolvimento segue a mesma linha de argumentação anterior, pois ele permite que o aprendiz realize sua aspiração de obter um programa executável, o que é gratificante e motivador. Aqui, a dificuldade para um iniciante é o fato de esses programas não serem tão lenientes com o usuário quanto aqueles aplicativos que o aprendiz costuma usar cotidianamente. Apesar de tudo, esse é o problema mais ameno com o qual se depara um iniciante em programação.

Para aliviar a sobrecarga cognitiva a que estarão submetidos os iniciantes em programação, alguns textos introdutórios usam linguagens criadas especificamente para ensino de programação e, provavelmente, a mais famosa delas seja Pascal. Contudo, além da notória carência de expressividade, essas linguagens incorporam pelo menos mais um contratempo que é difícil de ser superado: o que o aprendiz faz com tal linguagem depois que ela cumpre sua missão pedagógica?

A linguagem C tornou-se ao longo dos anos língua franca em programação, de modo que dominá-la é vital na formação de qualquer programador, mesmo que ele não pretenda se tornar especialista nessa linguagem, pois, entre outras razões, a sintaxe de C serve como base para a maioria das linguagens populares em uso corrente (p. ex., C++, Java, PHP, Perl e C#). Entretanto, um dos aspectos negativos de C é exatamente o fato de ela não ser muito palatável para ensino introdutório de programação. Diante desse arrazoado, o maior desafio deste livro consistiu em tentar simplificar o uso da linguagem C sem comprometer sua expressividade, de sorte que ela possa ser usada para demonstrar um número conveniente de técnicas de programação.

## Objetivos

O objetivo maior de um curso de introdução à programação deve ser ensinar o aprendiz a progredir desde o enunciado de um problema até sua resolução por meio de um programa. Tendo isso em consideração, este livro foi elaborado com o objetivo principal de ensinar programação utilizando uma pseudolinguagem para desenvolvimento de raciocínio algorítmico e a linguagem de programação C para concretização de algoritmos.

Os seguintes objetivos secundários também foram perseguidos na construção deste livro:

- Permitir acesso ao maior número possível de leitores, requerendo conhecimento mínimo de organização de computadores e nenhum conhecimento prévio de programação.
- Possibilitar a formação de programadores de competências medianas e capazes de resolver uma razoável gama de problemas de programação.
- Ensinar boas práticas de estilo de programação de modo que o aprendiz se torne capaz de construir programas que atendem critérios de legibilidade, manutenibilidade, robustez e portabilidade.
- Treinar programadores na frequente tarefa de encontrar e corrigir erros comuns de programação por meio de exposição de exemplos e exercícios sobre esses erros.

## Características do Livro

Este livro possui as seguintes características:

- Logo no primeiro capítulo, ele descreve passo a passo como construir programas monoarquivos simples em C utilizando o compilador GCC e o ambiente de desenvolvimento CodeBlocks, possibilitando que o leitor comece a programar desde cedo.
- Ensina construção de algoritmos usando uma pseudolinguagem simples e flexível e codificação usando as construções essenciais da linguagem C.
- Utiliza uma biblioteca, denominada LeituraFacil, que torna leitura de dados em C tão fácil quanto em uma linguagem algorítmica. Paulatinamente, o livro ensina como implementar essa biblioteca, que é 100% portável.
- A discussão sobre leitura de dados usando funções da biblioteca padrão, como **getchar**() e **scanf**(), que tanto frustram os iniciantes em programação em C, é adiada para um dos capítulos finais.
- Contém, distribuídos ao final de cada capítulo, cerca de 850 exercícios de revisão e 150 exercícios de programação. Os exercícios de revisão são separados por seções para facilitar a localização das respectivas respostas no texto. Além disso, são apresentadas respostas para a maioria dos exercícios de revisão bem como sugestões sobre como resolver a maioria dos exercícios de programação.
- Contém mais de 450 exemplos de programação exaustivamente comentados e devidamente testados.
- Enfatiza, desde o primeiro capítulo, o uso de um bom estilo de programação baseado em legibilidade, robustez, manutenibilidade e portabilidade.
- Discute princípios básicos de interação com usuário, notadamente interação dirigida por menu.
- Ressalta a prática de reúso de software, de modo que o leitor sinta-se motivado sabendo que o esforço despendido em seus primeiros programas será recompensado.
- Contém um índice que facilita sobremaneira a localização de informação distribuída no livro. Essa característica é especialmente importante na resolução de exercícios de revisão e programação.

Apesar de este livro ser aderente aos padrões C99 e C11, poucas novas características (p. ex., iniciadores designados) introduzidas por esses padrões mais recentes são usadas, pois a maioria delas (p. ex., suporte a programação concorrente) é considerada avançada demais para um iniciante.

### *Público Alvo*

Este livro destina-se primariamente a alunos de cursos da área de computação e informática que precisam aprender a programar como uma exigência fundamental em cursos de Ciência da Computação, Engenharia de Computação, Licenciatura em Computação, Engenharia de Software, Sistemas de Informação e cursos de graduação tecnológica dessa área, segundo recomendam as diretrizes curriculares do MEC em vigor. Este livro pode ainda ser utilizado por alunos de outros cursos de engenharia, como, por exemplo, Engenharia Eletrônica.

O material contido neste livro foi elaborado para um curso de um semestre de nível básico de programação e, naturalmente, requer como pré-requisito (ou co-requisito) conhecimentos básicos de organização de computadores. Idealmente, a carga horária recomendada é de 75 horas, mas o material dos **Capítulos 10** e **11** pode ser criteriosamente podado e o **Capítulo 12** pode ser refreado, de maneira que o material a ser exposto possa ser contido em uma disciplina com carga de 60 horas. Esse último capítulo é uma gentil introdução às estruturas de dados e pode ser adiado para uma disciplina de idêntica denominação. Por outro lado, o estudo completo de processamento de arquivos, apresentado no **Capítulos 10** e **11**, pode ser protelado para um curso mais avançado de programação.

Em virtude da facilidade de leitura, ao pouco conhecimento prévio requerido e aos inúmeros exemplos de programação, este livro também é adequado para autodidatas em programação.

## Organização do Livro

### *Capítulos*

O **Capítulo 1** começa com um histórico muito breve de linguagens de programação e prossegue apresentando uma visão geral de tradutores dessas linguagens. Em seguida, são discutidas as boas qualidades que devem nortear a construção de qualquer programa. O restante do capítulo tem caráter prático, pois é dedicado à instalação de um ambiente de trabalho. Aqui, descrevem-se as ferramentas utilizadas na construção de programas e ainda como funciona o processo de criação de um programa de console utilizando um ambiente integrado de desenvolvimento. Enfim, o **Capítulo 1** provê os fundamentos necessários para o acompanhamento do restante do texto.

O **Capítulo 2** é fundamental para uma disciplina introdutória, pois introduz o leitor à construção de algoritmos e à abordagem dividir e conquistar (ou de refinamentos sucessivos). Esse capítulo apresenta convenções de uma linguagem para construção de algoritmos e discute os operadores usados nessa linguagem e suas propriedades. O capítulo ainda descreve os conceitos de estrutura de controle e bloco de instruções e mostra a importância do uso de endentações e comentários na legibilidade de algoritmos. Finalmente, as etapas envolvidas na construção de um programa são descritas, com ênfase no papel desempenhado pelo algoritmo que representa o projeto do programa.

O **Capítulo 3** introduz a linguagem C e suas construções mais elementares, bem como expõe o conceito de código de caracteres. Nesse capítulo, são apresentados os tipos de dados primitivos **int**, **char** e **double** e as constantes que podem ser encontradas em um programa. Aqui, são descritos os operadores aritméticos, relacionais, lógicos, de incremento, de decremento e de atribuição. Os tipos de conversões que podem ocorrer em uma expressão também são discutidos. Além disso, esse capítulo expõe as regras sintáticas usadas na construção de identificadores, definições de variáveis, constantes simbólicas e comentários. Esse capítulo mostra quais são e como executar as etapas envolvidas na construção de um programa monoarquivo simples em C a partir de seu

algoritmo e introduz as bibliotecas padrão e LeituraFacil, que permitirão ao leitor criar programas interativos. As ferramentas usadas com esse propósito são o IDE CodeBlocks e o compilador GCC. O capítulo termina mostrando como o programador iniciante deve lidar com erros de sintaxe e advertências emitidas pelo compilador.

O Capítulo 4 descreve o fluxo natural de execução de um programa e como ele pode ser alterado com o uso de estruturas de controle. Esse capítulo expõe os conceitos de sequências de instruções e instruções vazias e examina os operadores condicional e vírgula. O Capítulo 4 acentua a distinção entre expressões condicionais e condições de parada em laços de repetição bem como apresenta uma discussão básica sobre geração de números aleatórios.

O Capítulo 5 enfoca definições, chamadas e alusões de funções. Esse capítulo começa introduzindo os conceitos básicos de endereços e ponteiros, que são essenciais para entender como se simula passagem de parâmetros por referência em C. Dois tópicos de grande importância expostos nesse capítulo são a classificação de parâmetros de acordo com seus modos e o que ocorre quando eles são passados durante uma chamada de função. Uma notação para representação de subprogramas em linguagem algorítmica também é proposta nesse capítulo, que encerra discutindo duração de variáveis e escopo de identificadores.

O Capítulo 6 discute várias práticas de estilo de programação que, se adotadas, evitarão vários dissabores decorrentes de erros de programação e da dificuldade de manutenção dos programas que o leitor irá construir. Esse capítulo explica a importância da documentação de funções e o uso prático de comentários com essa finalidade. Além disso, são discutidos alguns princípios básicos de interação com o usuário, em particular, interação dirigida por menus.

O Capítulo 7 é de natureza bastante pragmática, visto que discute erros de programação e apresenta duas técnicas elementares de depuração. Outro tópico de grande importância prática abordado nesse capítulo é o reúso de código e vários exemplos de reúso são apresentados aqui. Finalmente, esse capítulo lida com um tópico raramente visto em livros introdutórios, que é a imprecisão com que números reais são representados em computador.

O Capítulo 8 lida com o uso de arrays unidimensionais e ponteiros, bem como as relações existentes entre eles. Mas, antes disso, aritmética de ponteiros é cuidadosamente examinada. Definições e iniciações de arrays bem como o acesso a seus elementos são explorados nesse capítulo. Além disso, são apresentadas recomendações importantes sobre cuidados que devem ser adotados quando arrays são usados como parâmetros e retornos de funções e sobre o uso preventivo da palavra-chave **const**. Arrays multidimensionais são estudados apenas superficialmente, pois esse tópico pode ser adiado para um curso de programação mais avançado. Outros tópicos explorados nesse capítulo são o operador **sizeof** e os ameaçadores zumbis, que são variáveis de duração automática ou parâmetros que têm seus endereços retornados por funções.

O tema central do Capítulo 9 são caracteres e strings. Esse capítulo tenta antecipar e dirimir dúvidas comuns relativas ao uso de ponteiros, strings (constantes ou variáveis) e caracteres, que constituem fontes potenciais de confusão entre programadores iniciantes. As funções da biblioteca padrão de C para processamento de strings mais frequentemente utilizadas são aqui apresentadas. Esse capítulo mostra ainda como definir a função **main**(), de tal modo que um programa possa receber parâmetros quando ele for executado. O Capítulo 9 também descreve o módulo ctype dedicado à classificação e transformação de caracteres. Em virtude da importância dos temas abordados nesse capítulo, o material apresentado nele é bastante vasto e alguns exemplos são relativamente longos. Portanto espera-se que, se for o caso, o leitor ou instrutor seja capaz de discernir quais são os tópicos e exemplos mais relevantes a serem explorados.

Estruturas, uniões e enumerações são apresentadas no Capítulo 10, que também examina como o programador pode criar tipos de dados. Mais precisamente, esse capítulo mostra como definir e iniciar estruturas e uniões e como acessar os componentes dessas variáveis estruturadas. Os operadores de acesso são formalmente apresentados nesse capítulo, que mostra ainda como estruturas e uniões são usados como parâmetros e retornos

de funções. Além disso, esse capítulo examina o uso prático de uniões na construção de registros variantes. Enumerações são discutidas nesse capítulo, mas salienta-se que, ao contrário de estruturas e uniões, enumerações não são variáveis estruturadas. Além disso, esse capítulo discute iniciadores designados, que foram introduzidos pelo padrão C99.

O **Capítulo 11** define o conceito de *arquivo* tal qual ele é utilizado em C e discute o conceito e a implementação de streams nessa linguagem. Esse capítulo expõe os dois formatos básicos de streams: streams de texto e streams binários. Os significados de abertura e fechamento de arquivo e como essas operações são implementadas são discutidos nesse capítulo, que também descreve conceitos de entrada e saída comuns em programação, como buffers e streams padrão. Do ponto de vista mais prático, esse capítulo examina em profundidade leitura de dados via teclado utilizando funções da biblioteca padrão de C. Nesse capítulo, são ainda apresentadas as funções da biblioteca padrão de C mais frequentemente usadas em processamento de arquivos, bem como exemplos de uso de cada uma delas nas diversas formas de manejo de arquivos.

No **Capítulo 12**, os conceitos de alocação estática e dinâmica de memória são apresentados. Esse capítulo mostra como alocar e liberar memória dinamicamente por meio de chamadas de funções da biblioteca padrão de C. Esse capítulo também explora o tipo **void** * e a divisão em partições do espaço de execução de um programa. Em virtude de relativa exiguidade de tempo em uma disciplina introdutória, os tópicos desse capítulo podem ser estudados seletivamente ou adiados para uma disciplina mais avançada.

### *Apêndices*

O **Apêndice A** contém uma tabela com precedências e associatividades dos operadores da linguagem C usados neste livro, enquanto o **Apêndice B** é uma referência completa para a biblioteca **LeituraFacil**, mostrando como ela pode ser implementada e integrada nos ambientes de desenvolvimento suportados por esta obra. O **Apêndice C** apresenta respostas para a maioria dos **Exercícios de Revisão** de cada capítulo. O **Apêndice D** discute erros que são comuns em programação usando a linguagem C e é esncontrado exclusivamente online no site dedicado a este livro na internet (***www.ulysseso.com/ip***).

### *Programas Minimalistas*

**Programas minimalistas** são pequenos programas usados quando se pretende ilustrar uma característica específica de uma linguagem de programação. Mesmo que pareçam artificiais demais e corram o risco de não motivar o leitor, eles são inevitáveis em livros de programação.

Tentou-se testar por meio de programas minimalistas todas as construções da linguagem C, mas nem todos aspectos de uma dada construção foram testados. Assim, é importante usar o compilador como uma ferramenta de aprendizagem. Quer dizer, se uma determinada característica de alguma construção discutida neste livro não foi completamente compreendida, construa um pequeno programa que a teste e tire suas conclusões.

Programas minimalistas não são dirigidos para usuários comuns; i.e., eles devem ser usados exclusivamente por programadores (ou aprendizes de programação). Por isso, eles não seguem as recomendações de interação com o usuário preconizadas neste livro.

### *Exemplos de Programação*

Cada capítulo deste livro contém uma seção intitulada *Exemplos de Programação* na qual se pretendem demonstrar aplicações bem mais realistas e pragmáticas do que ocorre com programas minimalistas.

Programas de natureza prática são tipicamente longos demais para serem incluídos em um livro sem que o leitor esteja sujeito a perder o rumo. A abordagem adotada neste livro para inclusão dessa natureza é baseada em um princípio pedagógico (construtivismo) e em outro de engenharia de software (reúso de código). Isto é, usando

esses dois princípios, é possível apresentar programas longos e razoavelmente complexos. Mais exatamente, acredita-se que o uso de exemplos que usam parte de solução de outro problema e são desenvolvidos paulatinamente não apenas estimulam o aprendiz a valorizar o conceito de reúso de código, como ajudam a manter os programas relativamente curtos. Usando essa abordagem, é possível apresentar como exemplo um programa razoavelmente grande enfocando poucos conceitos e construções específicas da linguagem.

Alguns poucos exemplos de programação requerem conhecimentos específicos de uma determinada área de conhecimento e, quando isso ocorre, ao leitor é oferecido um **preâmbulo** cujo objetivo é prover definições e conceitos necessários para o completo entendimento do problema que será resolvido.

A maioria dos exemplos de programação é de natureza interativa, mas alguns são voltados para programadores e não para usuários comuns.

Alguns exemplos de programação são explicados em níveis de detalhes que talvez não sejam necessários para todos os leitores, mas aqueles menos experientes talvez precisem. Logo, o aprendiz pode ser seletivo na leitura das análises desses programas.

Comentários incluídos em programas foram escritos de modo informal em editores de programas (que, obviamente, não incluem corretores ortográficos) e não passaram pela competente revisão da editoria deste livro. Mesmo assim, nesses comentários, tentou-se ser o mais realista possível sem voluntariamente ofender o bom português.

### *Exercícios*

Ao final de cada capítulo são incluídos exercícios que servem para reforçar e ajudar a fixar o material exposto no respectivo capítulo. Esses exercícios estão divididos em duas categorias: Exercícios de Revisão e Exercícios de Programação.

- **Exercícios de Revisão**. Esses exercícios objetivam a verificação de aprendizagem de cada capítulo. Resolvendo os exercícios de revisão, o leitor pode fazer uma autoavaliação e, então, rever aquilo que ficou mal entendido. As respostas para a maioria das questões relacionadas a esses exercícios são encontradas no **Apêndice C**. Exercícios de revisão não requerem a escrita de programas, embora algumas questões possam ser respondidas prontamente desse modo. Exercícios sobre erros de programação devem merecer atenção especial por parte do leitor (v. adiante).
- **Exercícios de programação**. O objetivo desses exercícios é estritamente prático e eles requerem o uso de um computador e um ambiente de desenvolvimento adequados. A maioria dos exercícios de programação é acompanhada de sugestões que podem ser usadas para guiar as soluções para os problemas propostos. Deve-se salientar que não há solução única para cada problema e, além disso, pode-se constatar facilmente se uma proposta de solução é correta executando-se o programa (i.e., a solução) e verificando-se se os resultados são compatíveis com o enunciado do respectivo problema. O leitor deve tentar o maior número possível de exercícios de programação para aumentar sua confiança e motivação.

### *Rodapés*

Notas de rodapé referem-se a assuntos relacionados ao corpo principal do texto que não são de interesse de todos os leitores ou consistem de discussões adicionais que ajudam a clarificar os assuntos a que se referem. Quer dizer, essas notas podem ser consideradas conhecimento adicional e, se o leitor preferir, podem ser ignoradas sem prejuízo do conhecimento essencial que se pretende transmitir.

### Projetos de Programação

Ao final dos dois últimos capítulos são incluídos problemas classificados como *projetos de programação*. Esses problemas verificam a aprendizagem de boa parte do material apresentado neste livro e incorporam complexidades que demandam um tempo razoável para serem resolvidos. A solução de cada um desses problemas deve resultar em um programa de razoável dimensão.

Alguns problemas incluídos em seções intituladas *Exercícios de Programação*, notadamente nos capítulos de **8** a **12**, também podem ser classificados como projetos de programação dependendo da discrição de um instrutor experiente.

## Sequência de Estudo

A sequência de leitura deste livro é importante porque o material coberto em um capítulo depende do entendimento de capítulos anteriores, o material apresentado em uma seção pode depender do entendimento de uma seção anterior e assim por diante. Até mesmo um exemplo de programação pode depender de outro exemplo apresentado anteriormente no mesmo capítulo ou em um capítulo anterior.

Uma alternativa de estudo consiste em ler o material de cada capítulo sem se deter muito em cada tópico, estudar com atenção os exemplos e tentar resolver cada exercício proposto nas seções intituladas *Exercícios de Revisão* e *Exercícios de Programação*. Então, caso surjam dúvidas na resolução de algum exercício, o leitor deve retornar ao tópico referente ao exercício com o objetivo de dirimir suas dúvidas.

Em um livro sobre programação, é virtualmente impossível a apresentação de material de modo sequencial sem recorrer a assuntos que estão dispostos à frente na organização do livro. Assim, eventualmente, remete-se o leitor a uma seção mais adiante, mas isso não significa que conhecimento sobre o aludido assunto seja estritamente necessário para entendimento daquilo que está sendo correntemente exposto. Na maioria das vezes, uma simples leitura superficial do material mencionado é suficiente. Uma tal referência serve ainda para tranquilizar o leitor, pois ele terá nova oportunidade para esclarecer aquilo que está sendo discutido. Assim, será necessário interromper o fluxo de leitura para consultar tais seções, apenas se houver dificuldade para acompanhar aquilo que está sendo discutido, o que se espera que não ocorra com frequência.

## Material Complementar

### Hardware e Software

Este livro precisa ser complementado com um ambiente de laboratório. Praticamente, qualquer computador pessoal serve como hardware e o software é gratuito e facilmente encontrado na internet.

GCC é o compilador recomendado em qualquer plataforma e suportado por este livro, pois, além de produzir código de ótima qualidade, é gratuito. No sistema Mac OS X, também se pode usar GCC, mas, nesse caso, é mais recomendável usar o compilador Clang, que é mais moderno, oferece mensagens de erro mais compreensíveis e funciona de modo semelhante ao GCC. Sugestões de software de desenvolvimento para a plataforma Windows são apresentadas no **Capítulo 1**.

### A Biblioteca LeituraFacil

A biblioteca **LeituraFacil** foi desenvolvida pelo autor deste livro para o ensino introdutório de programação usando a linguagem C. Ela evita as dificuldades e frustrações com que se depara um iniciante em programação em C quando ele usa funções da biblioteca padrão de C para leitura de dados via teclado. A biblioteca **LeituraFacil** provê funções que tornam a leitura de dados via teclado tão simples quanto ela é numa linguagem algorítmica. Essas funções são 100% portáveis e, paulatinamente, o livro ensina como implementá-las.

No site dedicado ao livro na internet (v. adiante), são encontradas informações sobre instalação bem como o código-fonte dessa biblioteca, enquanto o **Apêndice B** discute essa biblioteca em detalhes.

### *Códigos-fonte*

Quase todos os exemplos de programas apresentados neste livro foram compilados usando o compilador GCC–MinGW versão 4.6.1 e testados nos sistemas operacionais Windows XP Service Pack 3 e Windows 7. Alguns poucos programas foram compilados com Borland C++ versão 5.0.2 (Windows XP), GCC 4.1.2 na distribuição Ubuntu 6.10 do sistema operacional Linux e Clang (Mac OS X) versão 4.0. Na ausência de informação explícita em contrário, o leitor deve assumir o uso do compilador GCC–MinGW e as respectivas versões de sistema Windows mencionados.

No site do livro na internet (v. adiante) encontram-se os códigos-fonte de todos os exemplos apresentados no texto, além de outros programas não inseridos nele. Esse material é classificado de acordo com os capítulos correspondentes no livro e encontra-se comprimido em formato *zip*. Enfim, você provavelmente não terá dificuldades para efetuar download utilizando qualquer navegador de web.

### *Site Dedicado ao Livro*

Este livro possui um site dedicado a si na internet que pode ser acessado no endereço: ***www.ulysseso.com/ip***. Esse site disponibiliza o seguinte material complementar:

- Código-fonte de todos os exemplos de programação apresentados neste livro bem como da biblioteca LeituraFacil
- Recomendações para instalação de ambientes de trabalho em plataformas que não são completamente cobertas pelo livro
- Links para importantes (e fidedignos) sites sobre programação na internet, onde se podem encontrar ferramentas de programação, códigos-fonte, artigos e outras referências úteis para aprender e aprimorar seu conhecimento sobre programação
- Errata
- Seção dedicada a críticas, sugestões e comentários

## Convenções Adotadas no Livro

### *Convenções Tipográficas*

Este livro utiliza *itálico* nas seguintes situações:

- Para enfatizar determinada expressão.
- Para representar componentes de construções da linguagem C ou de comandos de sistemas operacionais, compiladores ou linkers que devem ser substituídos por aquilo que realmente representam. Por exemplo, no comando do compilador GCC:

  ```
  gcc -c nome-do-arquivo
  ```

  *nome-do-arquivo* é um guardador de lugar que deve ser substituído por um verdadeiro nome do arquivo quando o comando for utilizado.
- Em palavras que representam estrangeirismos, exceto aquelas usadas intensamente em textos sobre programação ou computação em geral (p. ex., string, stream, linker).

Este livro utiliza **negrito** quando:

- Conceitos são definidos.
- Palavras-chaves e identificadores reservados da linguagem C aparecem no corpo do texto, mas esse 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.

Fonte **`monoespaçada`** é utilizada nos seguintes casos:

- Na apresentação de programa ou trechos (fragmentos) de programas.
- Na apresentação de comandos que aparecem em um console (terminal).
- Em nomes de arquivos e diretórios.
- Na representação de constantes numéricas, caracteres e strings.
- Na representação de identificadores que não são palavras-chave ou identificadores reservados.
- Na representação gráfica de teclas ou combinações de teclas. Nesse caso as teclas são escritas em maiúsculas e são colocadas entre colchetes. Por exemplo, **`[CTRL]+[Z]`** significa o pressionamento simultâneo das teclas rotuladas como *Ctrl* e *Z* num teclado convencional.

Fonte **`monoespaçada em negrito`** é utilizada para representar dados introduzidos por um usuário em exemplos de interação entre um programa e um usuário, enquanto a fonte `monoespaçada suave (light)` é utilizada para representar informações exibidas pelos programas.

A linguagem algorítmica usa a fonte **Acumin Pro**, sendo que as palavras-chaves dessa linguagem são **sublinhadas**.

Arquivos de cabeçalho que fazem parte da biblioteca padrão de C são colocados entre os símbolos < e > e escritos em **`fonte monoespaçada`**, como, por exemplo, **`<stdio.h>`**.

### Construções Sintáticas

Na descrição de construções sintáticas da linguagem C, as palavras em itálico representam guardadores de lugares. Por exemplo, no seguinte modelo sintático:

**while** ***(expressão)***
***instrução;***

*expressão* e *instrução* representam, respectivamente, uma expressão e uma instrução válidas da linguagem C, enquanto **while** é uma palavra-chave de C.

### Acentuação

O uso de acentuação infelizmente ainda é problemático em alguns consoles, notadamente naqueles da família Windows/DOS. Assim, para evitar que caracteres bizarros sejam exibidos na tela, foi decidido que os exemplos de programação não apresentassem palavras acentuadas no meio de saída padrão. Mas, é importante salientar que esse problema não é inerente à linguagem C. Por exemplo, mesmo empregando essa linguagem, programadores que usem um sistema operacional da família Unix que tenha sido devidamente localizado para a língua portuguesa não enfrentarão problemas dessa natureza.

Como é usual na maioria das linguagens de programação, identificadores são escritos sem acentuação ou cedilha. Novamente, essa não é uma limitação de C, pois, de acordo com os padrões C99 e C11, é permitido (mas bem complicado) criar identificadores acentuados. De qualquer modo, é aconselhável que o programador não se habitue a essa prática porque, como foi dito, a maioria das linguagens de programação não aceitam identificadores acentuados.

### Nomenclatura de Identificadores

As nomenclaturas utilizadas na escrita de cada categoria de identificadores (p. ex., variáveis, funções, tipos etc.) são descritas oportunamente ao longo do texto 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 parâmetros formais seguem as mesmas recomendações para nomes de variáveis.
- Nomes de rótulos de instruções seguem as mesmas convenções para escrita de variáveis, pois é praticamente impossível confundir um rótulo com uma variável.
- Nomes de tipos seguem as mesmas regras para nomes de variáveis, mas começam sempre com a letra *t*. Exemplo: `tPessoa`.
- Nomes de funções começam com letra maiúscula e seguem as demais regras para nomes de variáveis. Exemplo: `OrdenaLista`.
- Nomes de constantes simbólicas utilizam apenas letras maiúsculas; se um nome de constante simbólica for composto, utilizam-se subtraços para separar os componentes. Exemplos: `PI`, `TAMANHO_CPF`.

A principal vantagem da notação descrita acima é que ela requer o uso de subtraço apenas no caso de constantes simbólicas. Essa notação possui uma imensa legião de programadores que a seguem. Ela adota várias denominações e, talvez, a mais comum seja **notação de camelo** (as letras maiúsculas constituem as *corcovas do camelo*).

### Funções e Arrays

Como é comum em livros sobre programação em C e algumas outras linguagens, todas as referências a funções são seguidas de um par de parênteses [p. ex., **printf()**]. Essa convenção é adotada para evitar qualquer eventual ambiguidade, pois o nome de uma função considerado isoladamente representa seu endereço. Por uma razão semelhante, um array é referido usando-se seu nome seguido por colchetes (p. ex., `ar[]`).

### Três Pontos

Raramente, três pontos (`...`) são usados em fragmentos de programa para representar declarações e instruções omitidas por não serem relevantes na discussão em tela. Caso seja necessário usar algum desses fragmentos em um programa, os três pontos devem ser convenientemente substituídos (às vezes, pode-se simplesmente removê-los). Deve-se ressaltar, entretanto, que, quando três pontos são usados em protótipos e cabeçalhos de função com parâmetros variantes [p. ex., **scanf()**, **printf()**], eles assumem papel sintático.

### Simplificações

Para facilitar o ensino e a aprendizagem, a prática pedagógica precisa contar com simplificações. Aqui, essas simplificações não devem comprometer uma casual expansão futura de conhecimento em programação. Elas são descritas a seguir.

- **Tipos de dados primitivos**. Este livro usa apenas os tipos primitivos: **int**, **char**, **double** e **void**. De acordo com o padrão ISO C11, existem nove tipos inteiros primitivos, três tipos de ponto flutuante reais e seis três tipos de ponto flutuante complexos. Apesar dessa abundância de tipos, raramente, na prática, eles são usados em um programa, seja por falta de portabilidade (p. ex., **int**), seja por falta de necessidade (p. ex., **long double _Complex**). Assim, a escolha dos tipos usados neste livro introdutório parece adequada, pois evita a discussão sobre um assunto relativamente supérfluo.
- **Número real versus número de ponto flutuante**. Não existem números reais em computação e esse fato deve ser aprendido em qualquer curso básico de organização de computadores. Alguns textos

tentam chamar atenção para esse fato denominando números que representam aproximadamente números reais como números de ponto flutuante. Mas, há dois problemas com essa consideração: um conceitual e outro pedagógico. Do ponto de vista conceitual, não é verdade que números reais sempre são representados como números de ponto flutuante, apesar de essa ser a forma mais moderna e comum de representação desses números. Além disso, essa denominação confunde abstração (número real) com implementação (ponto flutuante). Do ponto de vista pedagógico, a denominação número de ponto flutuante requer o esclarecimento de conceitos que não são essenciais num texto introdutório de programação.

- **Compilação versus construção de programa executável**. Compilar um arquivo-fonte não é o mesmo que transformá-lo em programa executável. Essa distinção é claramente apresentada no **Capítulo 3**. Mesmo assim, como é comum no jargão de programação, algumas vezes, *compilar um programa* é utilizado com o significado de *construir um programa executável*. Espera-se que o leitor seja capaz de deduzir do contexto o real significado de *compilação*.
- **Stream versus arquivo**. Stream é um conceito utilizado por C e outras linguagens de programação que permite que se processem arquivos sem dar atenção ao dispositivo usado como origem ou destino de dados. Arquivo, por sua vez, representa qualquer dispositivo do qual se podem ler dados ou no qual se podem depositá-los. Apesar de muitos leitores estarem familiarizados com o conceito mais usual de arquivo (p. ex., uma coleção de bytes armazenada em disco rígido), eles não aparentam ter dificuldade em entender esse conceito generalizado de arquivo. Enfim, o termo *arquivo* é muitas vezes utilizado onde o termo mais adequado deveria ser *stream*. O **Capítulo 10** esclarece melhor essa aparente ambiguidade.
- **Unix versus Linux**. *Linux não é Unix* é uma frase proferida *ad nauseum* na internet, mas, do ponto de vista das menções apresentadas neste livro não há nenhuma diferença entre esses sistemas.
- **Indireção**. Essa palavra inexiste no vernáculo, mas, mesmo assim, é usada para representar o ato de acessar o conteúdo de um bloco de memória utilizando um ponteiro. Isto é, indireção significa acesso *indireto* a um conteúdo em memória por meio de um ponteiro, em vez de via acesso direto promovido por nomes de variáveis.
- **Subtraço**. Na língua portuguesa, esqueceram de rotular o traço que aparece em teclados convencionais acima do traço usado em hifenação e como sinal de menos em Matemática. Em inglês, este caractere é denominado *underscore* ou *underline*. Neste livro, este símbolo será denominado *subtraço*, mas esse vocábulo não existe oficialmente no vernáculo.
- **Parâmetro versus argumento**. No jargão da linguagem C, *parâmetro* é o termo utilizado para representar o conceito que, neste livro, é denominado *parâmetro formal*, enquanto *argumento* nesse jargão é aqui chamado *parâmetro real*. Várias outras linguagens de programação usam a terminologia adotada aqui. Neste texto, *argumento* é usado apenas como referência a um string recebido por um programa via linha de comando.
- **Parâmetro qualificado com const**. Só faz sentido qualificar um parâmetro com **const** quando ele é um ponteiro e, nesse caso, a qualificação se refere ao conteúdo para o qual o parâmetro aponta e não ao parâmetro em si. Do ponto de vista pragmático, não faz sentido qualificar com **const** um parâmetro que não é ponteiro.
- **Array como parâmetro 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 parâmetro* ou *retorna um array*, o que se quer dizer é, respectivamente, que a função *recebe o endereço de um array* ou *retorna o endereço de um array*. A mesma simplificação aplica-se a strings, que, a propósito, também são arrays.

- **Leitura de strings**. Nenhuma função, quer ela faça parte da biblioteca padrão ou LeituraFacil, é capaz de ler um string introduzido pelo usuário por uma singela razão: é impossível para um usuário digitar o caractere nulo que faz parte de qualquer string em C. Assim, neste texto, *ler um string* significa ler caracteres, armazená-los num array e acrescentar o caractere nulo, de modo que o resultado armazenado seja um string.
- **Leitura de valores inteiros e reais via teclado**. A única informação que pode ser lida via teclado são caracteres, pois o stream padrão que o representa é um stream de texto. Portanto *ler um inteiro* significa precisamente ler caracteres que possam eventualmente ser convertido num número inteiro. O mesmo raciocínio aplica-se a leitura de valores reais.
- **Precedência dos operadores de incremento/decremento sufixos**. De acordo com os padrões ISO C99 e C11, a precedência dos operadores de incremento e decremento sufixos é a mesma precedência dos operadores binários de acesso, que fazem parte do grupo de operadores com a maior precedência da linguagem C. Neste livro, todos os operadores unários (incluindo os operadores de incremento/decremento sufixos) são considerados integrantes de um mesmo grupo de precedência, como na especificação original da linguagem C. Essa simplificação não compromete o raciocínio que norteia o uso desses operadores. Por exemplo, na expressão:

  ```
  *p++
  ```

  de acordo com o padrão C11, o operador `++` é aplicado primeiro porque ele tem precedência maior do que do o operador de indireção. Neste livro, chega-se à mesma conclusão, mas ela é fundamentada na associatividade dos operadores unários, que é à direta (e o operador `++` está à direita do operador `*`).
- **Reúso de Código**. Neste livro, reúso de código é usado no sentido literal e não como em disciplinas de engenharia de software, nas quais reúso de software refere-se ao design de componentes de modo a facilitar o reúso deles. Neste livro introdutório, funções não são projetadas levando a possibilidade de reúso em consideração. Aqui, reúso de código refere-se apenas ao uso de programas ou algoritmos eventualmente já existentes na construção de novos programas ou algoritmos.

## *Práticas Inimitáveis*

Este livro adota práticas que o programador deve evitar em sua profissão, mas que em um livro didático são justificáveis, como se alega a seguir:

- **Comentários didáticos**. Na prática, comentários devem ser escritos para programadores que conhecem a linguagem utilizada e não têm caráter didático, como a maioria dos comentários apresentados no texto. Naturalmente, os comentários apresentados aqui são didáticos porque estão inseridos num *livro didático*.
- **Números mágicos**. Em vários trechos deste livro recomenda-se que números mágicos sejam substituídos por constantes simbólicas. Mesmo assim, números mágicos abundam em exemplos de expressões e definições de arrays. A justificativa para esse procedimento é que esses exemplos não constituem programas completos e, sendo assim, não existe contexto suficiente para que se encontrem nomes significativos para associar a esses números mágicos.
- **Verificação de valores retornados por funções**. Muitas 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 esses valores, em vez de assumir que uma determinada operação sempre obtém êxito (v. *Lei de Murphy* no **Capítulo 11**). Em alguns poucos exemplos apresentados no texto, esses valores não são testados apenas para não desviar atenção daquilo que um exemplo pretende enfocar. Outras vezes, esses valores não são testados exatamente para mostrar consequências danosas decorrentes dessa prática.

## Organizadores Prévios

No início de cada capítulo é incluído um quadro contendo as competências que o leitor deverá adquirir após o estudo desse capítulo. Esses quasdros são denominados **organizadores prévios** e foram criados pelo psicólogo educacional americano David Ausubel (v. **Bibliografia**). Segundo seu criado, um organizador prévio facilita o entendimento do novo material que será apresentado no capítulo.

## Críticas, Sugestões e Comentários

Apesar de a maior parte do material já ter sido utilizada durante muitos anos em forma de notas de aula e, consequentemente, ter sido submetida a inúmeras correções, sua expansão inevitavelmente introduziu erros ou imperfeições que não puderam ser detectadas e corrigidas em tempo. Desse modo, o autor se desculpa por qualquer deslize e aceita qualquer crítica ou indicação de erros encontrados no livro.

Se, porventura, algum programa encontrado no livro ou no site dedicado a ele apresentar um comportamento inesperado e o leitor acreditar tratar-se de um erro de programação, pode entrar em contato com o autor. Também, qualquer questão de natureza técnica ou comentário útil pode ser enviado usando formulário próprio no site do livro.

## Agradecimentos

Este livro é oriundo de notas de aula continuamente refinadas durante vários semestres de ensino da disciplina Introdução à Programação no curso de Ciência da Computação da Universidade Federal da Paraíba. Desse modo, o autor gostaria de agradecer a alunos, monitores e professores do Departamento de Informática da UFPB que indicaram falhas em versões anteriores do texto e apresentaram sugestões para sua melhoria.

*Ulysses de Oliveira*

*Abril de 2014*