Prefácio

Ao Leitor

A Importância de C

A linguagem C foi criada no início dos anos 1970, mas, apesar da idade, parece ser uma linguagem que nunca envelhece, pois, ainda mantém seu lugar garantido entre as linguagens de programação mais populares. Eis algumas razões que justificam a importância de saber programar em C:

- A linguagem C é considerada excelente para programação de sistemas e os sistemas operacionais mais utilizados atualmente são escritos em C ou C++. Em particular, é necessário saber programar em C para ser capaz de adequar às suas necessidades o *kernel* e outros componentes do sistema operacional Linux.
- Em termos de desempenho, programas bem escritos em C só podem ser comparados aos equivalentes escritos na linguagem assembly. Estima-se que um programa em C seja apenas de 1% a 2% mais lento do que um programa escrito em assembly para uma determinada máquina. Mas, programar em assembly é bem mais difícil do que em C, além de produzir programas sem portabilidade.
- Nem todo programador escreve aplicativos com agradáveis interfaces gráficas ou programas para gerenciamento de banco de dados. Isto é, muitos programadores escrevem programas para sistemas embutidos (ou embarcados), como, por exemplo, um programa para controle de um forno de microondas. Pode-se usar C ou uma linguagem próxima de C (por exemplo, *SystemC*) para esta finalidade.
- C serve como base para pseudo-linguagens nas quais muitos livros sobre algoritmos e estruturas de dados descrevem algoritmos. Portanto, qualquer programador versado em C sente-se mais à vontade lendo estes livros.
- Dada a predominância de C como linguagem de programação durante um longo período, existe grande possibilidade de um programador herdar boa porção de código legado escrito em C para efetuar manutenção.
- Apesar de todas as críticas negativas relacionadas à sua sintaxe intricada, C tem servido de base para a maioria das linguagens populares em uso corrente: C++, Java, PHP, C# e outras. Esse fato sugere que, sabendo-se C, pode-se migrar mais facilmente para outras linguagens mais modernas nela baseadas.
- Entender C em contraposição a outras linguagens é semelhante a entender Unix (ou Linux) em contraposição a sistemas operacionais da família Microsoft Windows. É notório que usuários de Unix/Linux entendem muito mais sobre o

funcionamento de um sistema operacional do que usuários de Windows. De modo análogo, programadores de C entendem muito mais do funcionamento do computador do que programadores de outras linguagens relativamente mais fáceis de usar. Perceber o que acontece em segundo plano enquanto um programa está sendo executado é crítico para entender o funcionamento de programas escritos em qualquer linguagem. Este conhecimento ainda é importante nos dias atuais, mesmo que o interesse em otimização e flexibilidade tenha diminuído.

- O mais recente padrão ISO da linguagem C, popularmente conhecido como C99, introduz características importantes que não estão presentes nem mesmo em linguagem consideradas mais modernas e avançadas, como C++ e Java.

Os argumentos apresentados acima não significam que C seja melhor do que qualquer outra linguagem. Isto é, nenhuma delas é perfeita para todas as tarefas de programação e, na escolha de uma linguagem, não se devem deixar de levar em conta outras considerações tais como:

- A rapidez com que se cria um programa depende muito da experiência do programador na linguagem e do código disponível que pode ser utilizado na construção do programa.
- À medida que o desempenho dos computadores cresce, a importância de escrever código mais eficiente diminui. Portanto, utilizar C apenas devido à eficiência de código vem cada vez mais perdendo a tradicional relevância.
- Linguagens mais restritivas do que C (Java, por exemplo) reduzem a possibilidade de surgimento de erros de programação, apesar de ser menos flexíveis.
- Além de um programa em C estar mais propenso a erros de programação, também é mais difícil torná-lo seguro. São famosos os casos de violação de sistemas devido a descuidos no uso de funções de entrada e saída formatadas, como **scanf()** e **printf()**.

Portanto, é necessário levar esses aspectos em consideração antes de tomar uma decisão sobre a linguagem a ser utilizada em desenvolvimento de software.

Público-alvo

Este livro foi elaborado com dois objetivos gerais: (1) ensinar programação utilizando a linguagem de programação C e (2) servir como referência para programadores que usam esta linguagem. O livro destina-se primariamente a professores e estudantes de cursos da área de computação e informática. O material coberto nos dois volumes do livro constitui uma disciplina fundamental nos cursos de Bacharelado em Ciência da

Computação, Engenharia de Computação, Licenciatura em Computação, Sistemas de Informação e cursos de graduação tecnológica da área de computação e informática, segundo recomendam as diretrizes curriculares do MEC em vigor. O primeiro volume pode ainda ser utilizado em disciplinas de alguns outros cursos de engenharia, como, por exemplo, Engenharia Eletrônica. Além disso, os dois volumes do livro poderão ser utilizados por administradores de sistemas da família Unix, participantes de cursos de programação de curta duração e autodidatas em programação.

O material contido no **Volume I** foi elaborado para um curso de um semestre de nível intermediário de programação utilizando a linguagem C. Seu uso é recomendado idealmente como segundo curso de programação quando o aluno já tem conhecimento básico de organização de computadores e domínio de construção de algoritmos. Portanto, se o livro for utilizado como um curso introdutório de programação, ele deve ser complementado com um texto que preencha estes pré-requisitos.

Feitas essas ressalvas, o livro pode ser utilizado por alunos não apenas da área de computação como também de outras áreas afins. Quando utilizado por alunos de computação, todo o conteúdo do **Volume I** pode ser abrangido numa disciplina de 60 horas/aula. Como livro-texto para cursos de outras áreas, alguns tópicos mais avançados (como, por exemplo, programação de baixo nível) podem ser omitidos.

O livro possui as seguintes características que, tanto quanto se sabe, não são encontradas em outros textos nacionais similares sobre o assunto:

- Apresenta todas as novidades introduzidas pelo padrão ISO/IEC 9899:1999 (popularmente conhecido como C99), incluindo a Correção Técnica 1 (*Technical Corrigendum 1*), publicada em 2001, e a Correção Técnica 2 (*Technical Corrigendum 2*), publicada em 2004. Portanto, acredita-se que este livro seja um dos mais atuais da literatura de programação.
- Explora completamente todos os módulos da biblioteca padrão de C (**Volume II**).
- Todos os exemplos apresentados foram testados pelo autor e por monitores supervisionados pelo autor em compiladores que satisfazem o padrão C99.
- Contém exercícios propostos de revisão do assunto e de programação ao final de cada capítulo. A única exceção é o **Capítulo 1**, que não possui exercícios de programação.
- Descreve passo a passo a construção de programas simples (**Capítulo 2**) e multiarquivos (**Capítulo 4**) em C utilizando o compilador gcc, o ambiente DevC++ e o utilitário make.
- Introduz depuração de programas passo a passo usando o depurador gdb (**Capítulo 6**).

- Introduz operações de entrada/saída básicas logo no **Capítulo 2**, de modo que, seguindo sequencialmente o material, o aluno pode começar a programar dentro de duas ou três semanas do início do curso numa disciplina com oito horas semanais.
- Induz o uso de bom estilo de programação em C, apresentando sugestões claras sobre como desenvolver um estilo próprio aderente à maioria dos manuais de estilo de programação.
- Contém um capítulo (**Capítulo 13**) específico sobre programação de baixo nível em C, que é um dos tópicos que despertam maior interesse nesta linguagem.

Escopo do Livro

Muitos alunos sentem-se frustrados quando percebem que, em muitos cursos de programação, eles não aprendem a construir programas com interfaces gráficas e interação dirigida por eventos como a maioria dos programas que costumam utilizar. Contudo, programação de interfaces gráficas envolve conceitos e técnicas (como uso de cores, fontes, layout) que, além de acrescentarem complexidade, pouco têm a ver com programação em si. Estar envolvido simultaneamente com estes conceitos enquanto se aprende a programar é um fator de distração do objetivo principal.

Este livro ensina a programar utilizando a linguagem C num ambiente baseado em texto e não se propõe a ensinar programação de interfaces gráficas ou dirigida por eventos. Todavia, o conhecimento a ser explorado é necessário para construir interfaces gráficas mesmo quando se tem disponível um ambiente de desenvolvimento rápido (RAD) que gera boa parte do código automaticamente.

Este livro utiliza conhecimento relativo à construção de algoritmos, arquitetura de computadores e sistemas operacionais apenas na medida em que estes assuntos se fazem necessários dentro de determinados contextos. O aprofundamento nestes temas está, portanto, fora do escopo do livro.

O padrão ISO C99 define dois tipos de sistemas: (1) **sistemas com hospedeiro** e (2) **sistemas livres**. Sistemas livres não possuem hospedeiro (por exemplo, sistema operacional) ou sistema de arquivos e não precisam atender a todas as recomendações impostas para sistemas com hospedeiro. Por exemplo, o padrão ISO não requer que estes sistemas dêem suporte completo à biblioteca padrão de C. Aqui, abordam-se apenas sistemas com hospedeiro (mais precisamente, programas executados sob a supervisão de um sistema operacional), mas também pode servir de base para sistemas livres (por exemplo, programas embutidos).

Organização do Livro

Com o acréscimo na linguagem C, notadamente em sua biblioteca padrão, promovido pelo novo padrão ISO C99, tornou-se difícil expor em profundidade todo o assunto num único volume. Assim, decidiu-se dividir o material em dois tomos. O corrente volume expõe todos os aspectos fundamentais de C e o conteúdo de cada capítulo dele será descrito a seguir. Logo depois, será apresentada uma sinopse do **Volume II**.

Volume I

O **Capítulo 1** apresenta as construções básicas da linguagem C, como seus tipos de dados elementares e suas estruturas de controle. Um tópico fundamental apresentado neste capítulo são as propriedades dos operadores de C. O completo entendimento destas propriedades e suas conseqüências na avaliação de expressões são fundamentais para o bom acompanhamento do restante do material. Este capítulo é denso em exposição de conceitos e escasso em aplicações práticas, de modo que o leitor pode sentir-se entediado. Os tópicos relativos a conversões automáticas e tipos complexos, dependendo da ênfase do curso, podem ser estudados apenas superficialmente. O estudo deste capítulo pode ser intercalado com o estudo do **Capítulo 2**, de modo que o aprendiz possa começar a praticar programação em C o mais cedo possível.

O **Capítulo 2** tem conteúdo essencialmente prático que ensina como construir programas simples em C (i.e., programas que consistem de um único arquivo), tanto utilizando um ambiente integrado de desenvolvimento (IDE) quanto utilizando um editor de texto em conjunto com um compilador executado em linha de comando. Apesar de ser comum ter-se um programa escrito em C que não interage diretamente com o usuário (por exemplo, um programa que implementa um serviço *daemon*), tipicamente, aprende-se a programar construindo-se programas interativos simples, onde o usuário é o próprio programador. Assim, o capítulo apresenta as funções mais elementares de entrada e saída, bem como o modo aproximadamente correto de utilizá-las. É fundamental entender o funcionamento da entrada de dados padrão já que este, provavelmente, será o meio de entrada mais utilizado enquanto se pratica programação em C.

O **Capítulo 3** basicamente explora a definição e uso de funções. Este capítulo começa introduzindo os conceitos fundamentais de endereços e ponteiros que são essenciais para entender como se pode simular passagem de parâmetros por referência em C, que, a rigor, possui apenas passagem de parâmetros por valor. Do ponto de vista prático, são apresentados dois tópicos essenciais para a construção de programas interativos amigáveis: leitura e validação de dados e interação dirigida por menus. Funções recursivas, *inline* e com listas de argumentos variáveis também são apresentadas neste capítulo.

O **Capítulo 4** ensina a construir programas multiarquivos. Aqui são discutidos conceitos importantes, como classes de armazenamento, tipos de dados derivados e qualificadores de tipos. Do ponto de vista funcional, descreve-se o uso de ambientes de desenvolvimento e do utilitário make na construção de programas distribuídos em múltiplos arquivos. Devido à sua importância prática, este capítulo deve ser cuidadosamente examinado.

O **Capítulo 5** introduz o pré-processador de C e suas construções próprias. Entender a diferença entre instruções da linguagem C e instruções do pré-processador de C (i.e., diretivas) é crucial. Os sub-tópicos fundamentais deste capítulo são macros com argumentos e a inclusão de arquivos. Conhecer as diretivas **#error**, **#pragma** e **#line** é útil, mas não é essencial. Diretivas de compilação condicional são de interesse mediano.

O **Capítulo 6** apresenta legibilidade e depuração de programas e estes temas não devem ser negligenciados. A linguagem C, em virtude de suas construções muitas vezes quase criptográficas, requer cuidados especiais na adoção de um estilo de programação que garanta grau de legibilidade aceitável. Por exemplo, se um programa em C não for formatado consistentemente e se identificadores muito abreviados forem usados, ele será muito difícil de entender. Assim, neste capítulo são apresentadas sugestões importantes sobre como tornar os programas mais legíveis e fáceis de ser mantidos. Outro tópico importante e de natureza prática, raramente visto em outros livros de programação, é depuração de programas. Neste capítulo, são apresentadas técnicas elementares de depuração, como o uso preventivo do compilador e o emprego de **printf()**, comentários, asserções e compilação condicional. O uso adequado de depuradores de alto-nível e, em particular, o uso do depurador gdb são explorados neste capítulo.

O **Capítulo 7** enfoca arrays e ponteiros, bem como as relações entres eles. Aconselha-se cautela a leitores no estudo deste capítulo, principalmente nos tópicos relacionados a aritmética de ponteiros, visto que este assunto contém algumas sutilezas e, ao mesmo tempo, é essencial para o domínio da linguagem C. Várias novidades introduzidas pelo padrão C99, como arrays com índices estáticos e ponteiros restritos, são também expostas neste capítulo. O tópico relacionado a arrays multidimensionais pode ser estudado apenas superficialmente ou completamente omitido, dependendo da ênfase do curso ou de disponibilidade de tempo.

O tema central do **Capítulo 8** são os *strings*. As funções da biblioteca padrão de C para processamento de *strings* mais comumente utilizadas são aqui apresentadas, enquanto que outras funções desta categoria são exploradas no **Volume II**. Este capítulo mostra ainda como definir a função **main()**, necessária em programas hospedados, de tal modo que um programa possa receber parâmetros quando ele for executado. O capítulo é concluído com uma seção que analisa problemas associados com leitura de *strings* realizada por funções da biblioteca padrão e propõe um procedimento prático que supera estes problemas. Este último tópico pode ser considerado opcional.

Estruturas, uniões e enumerações são apresentadas no **Capítulo 9**. Novas construções introduzidas em C99, como iniciadores designados, arrays flexíveis e literais compostos estão incluídos aqui. Atenção especial deve ser dispensada às enumerações, visto que este tipo de dado constitui assunto muitas vezes negligenciado na maioria dos cursos de programação.

O **Capítulo 10** intitula-se construções complexas por discutir algumas das construções da linguagem C mais complicadas de entender. Todas estas construções envolvem o uso de ponteiros: arrays de ponteiros, ponteiros para ponteiros e ponteiros para funções. Apesar de importante para um completo domínio da linguagem C e para alunos de cursos nos quais programação é uma disciplina essencial, este capítulo pode ser preterido, dependendo da disponibilidade de tempo e da ênfase do curso.

No **Capítulo 11**, alocação dinâmica de memória é apresentada. O conhecimento deste tópico é imprescindível na formação de qualquer programador. Neste capítulo, também são introduzidas listas encadeadas e suas operações fundamentais. Pode-se deixar este último tópico para ser explorado num curso subseqüente de estruturas de dados, mas, assim procedendo, dificilmente, o aprendiz conseguirá realizar a utilidade prática de alocação dinâmica de memória.

O **Capítulo 12** expõe um dos tópicos mais importantes e complexos em programação: entrada e saída de dados. Este capítulo é intitulado *Processamento de Arquivos* em alusão à definição generalizada de arquivo utilizada tanto em C quanto em sistemas operacionais da família Unix. Conceitos de entrada e saída comuns a programação em outras linguagens, como *streams* e buffers, são discutidos aqui. São apresentadas ainda todas as funções para processamento de arquivos encontradas no módulo stdio da biblioteca padrão de C juntamente com exemplos de uso de cada uma delas. É preciso muita atenção para algumas sutilezas que muitas vezes passam despercebidas quando se lida com processamento de arquivos em C. Funções para processamento de entrada e saída de caracteres extensos e multibytes encontradas no módulo wchar da biblioteca padrão de C são apresentadas no **Volume II**.

O **Capítulo 13** faz uma introdução à programação de baixo nível em C e requer, por parte do leitor, conhecimentos básicos de organização e arquitetura de computadores. Algumas aplicações práticas, como o uso de sinalizadores e criptografia, são apresentadas aqui. Os tópicos expostos podem ser considerados avançados para um curso básico ou intermediário em programação e podem ser explorados num curso mais avançado ou, por exemplo, como introdução a um curso de programação de sistemas embutidos (ou embarcados).

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 de entrada e saída das

famílias de funções printf e scanf. *Strings* de formatação usados por estas funções são examinados em profundidade no **Volume II**. Estes apêndices podem ser utilizados como fonte freqüente de referência durante a construção de programas em C.

Poucos exemplos apresentados no livro constituem programas completos e, quando ocorre, eles são muito pequenos. Tenta-se manter cada exemplo tão simples quanto possível com o objetivo de enfocar um ou alguns poucos aspectos práticos da linguagem C de cada vez. Uma completa discussão de programas longos prejudicam o fluxo de apresentação do material e não existe evidência de que a leitura de longos programas melhore o aprendizado. Por outro lado, de acordo com a experiência acumulada ao longo de vários cursos de programação, detectou-se que o aluno fica mais motivado quando consegue entender todos os detalhes de um programa simples do que quando toma conhecimento dos possíveis problemas que podem ser resolvidos, sem entender completamente como os programas correspondentes funcionam. Devido às dimensões dos exemplos, vários deles podem ser completamente explorados e absorvidos pelo aluno numa única aula.

Ao final de cada capítulo são incluídos exercícios divididos em duas categorias: (1) **Exercícios de Revisão** e (2) **Exercícios de Programação**.

Os **Exercícios de Revisão** objetivam a verificação de aprendizagem de cada capítulo. Resolvendo os exercícios propostos, o leitor pode fazer uma auto-avaliação e rever aquilo que ficou mal entendido ou preencher lacunas detectadas em seu conhecimento.

O objetivo dos **Exercícios de Programação** é estritamente prático. Espera-se que o leitor resolva-os num laboratório contendo um ambiente de desenvolvimento adequado. A ordenação dos exercícios segue grau de dificuldade crescente e, quando apropriado, são apresentadas sugestões para problemas considerados mais complexos.

Não são incluídas soluções para os exercícios propostos por duas razões. No caso dos **Exercícios de Revisão**, as respostas são encontradas diretamente no texto ou, indiretamente, usando-se um mínimo de dedução. No caso dos **Exercícios de Programação**, 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.

Em um livro sobre programação, é virtualmente impossível a apresentação de material de modo seqüencial sem recorrer a assuntos que estão à frente na organização do livro. Nestes casos, 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.

Finalmente, existem muitas comparações de C com outras linguagens de programação ao longo do texto. Estes comentários refletem a experiência do autor no ensino de diversas disciplinas de linguagem de programação. Se você desconhece alguma das linguagens mencionadas, simplesmente, ignore tais comparações pois elas não comprometem o completo aprendizado de C.

Volume II

Todos os componentes (tipos, macros, funções e variáveis globais) dos 24 módulos que integram a biblioteca padrão de C são minuciosamente examinados no **Volume II**. O estudo destes módulos domina a maior parte deste volume, mas outros assuntos importantes, como localização de programas, portabilidade de programas e processamento de caracteres extensos e multibytes, também são explorados.

O **Volume II** contém ainda apêndices que podem servir como referência para programadores e aprendizes da linguagem C. Estes apêndices apresentam como conteúdos: resumo das palavras reservadas da linguagem C, composição de *strings* de formatação de entrada e saída, resumo das alterações introduzidas pelo padrão C99 e uma lista de erros comuns de programação em C.

Como Usar o Livro

Recomendações ao Aprendiz

Nenhuma linguagem de programação pode ser aprendida simplesmente estudando-se material escrito. Programação é algo que se aprende apenas com muita prática e perseverança. A situação é análoga ao aprendizado de algumas disciplinas (por exemplo, Física e Matemática) que requerem a resolução de muitos problemas antes que se obtenha um completo domínio de uma metodologia de resolução de problemas. Algumas sugestões para um melhor aprendizado das técnicas de programação apresentadas aqui são:

- Edite e execute os exemplos apresentados no texto, tentando entender como estes funcionam. Arquivos contendo estes exemplos podem ser obtidos via internet (v. mais adiante).
- Faça modificações nos exemplos apresentados e tente responder questões associadas a elas. Por exemplo, *O que acontece quando eu troco uma instrução por outra análoga? Se eu alterar o programa, ele continuará funcionando satisfatoriamente? O desempenho do programa melhora com esta mudança? Essa mudança torna a solução mais legível*? Quando um programa deixar de

funcionar, tente entender o porquê. Testando suas próprias idéias a respeito de um dado programa você certamente obterá um melhor entendimento sobre como funcionam programas escritos em C.

- Um bom depurador constitui uma excelente ferramenta de aprendizagem de programação. Utilize um depurador para executar seus programas mesmo quando um dado programa não precisa ser depurado e você aprenderá muito sobre o funcionamento de programas e do próprio computador. Esta sugestão é detalhada no **Capítulo 6**.
- Desenvolva desde cedo um estilo de programação. Assim como não existe um estilo único de escrita de redações, também não existe um estilo único de escrita de programas. Mas, se você aderir aos conselhos básicos de estilo de programação apresentados no livro, seu estilo estará bem fundamentado.
- Tente resolver todos os exercícios de programação propostos ao final de cada capítulo.

Com relação aos exercícios de programação, é recomendável que você os resolva individualmente. Embora você seja encorajado a discutir ou trabalhar com outros colegas na elaboração de uma solução para um exercício de programação, sugere-se que a implementação de cada solução seja individual. Em geral, é crucial que você apresente não apenas um programa que simplesmente funcione, mas leve ainda em consideração outros critérios tais como estrutura, simplicidade e clareza, que classifiquem seu programa como um produto de qualidade. O **Capítulo 6** deste livro apresenta várias sugestões neste sentido. Um bom programa reflete um bom projeto de solução para o respectivo problema. Portanto, pense bastante sobre a linha de solução (algoritmos) a ser seguida antes de começar a codificar seu programa.

O material complementar que o aluno deve ter disponível para acompanhamento deste texto consiste de um computador e um ambiente de programação C. Sugestões sobre como obter este material complementar são apresentadas mais adiante.

Por último, apesar de o aspecto prático ser fundamental, você não deve relegar a parte conceitual a segundo plano, pois, caso contrário, mesmo que se torne um programador habilidoso, não terá plena consciência do que está fazendo.

Recomendações ao Instrutor

Do ponto de vista conceitual, recomenda-se ao instrutor que a maior ênfase seja destinada aos aspectos semânticos e pragmáticos em detrimento a aspectos sintáticos da linguagem C. Portanto, ressalte o funcionamento das estruturas de controle, dos operadores, de passagens de parâmetros e o papel desempenhado pelos diversos tipos de dados e funções. Aconselhe os alunos a adotarem um estilo próprio de codificação,

chamando atenção para o fato de não haver um único estilo correto, mas que existem regras unânimes, como representatividade de identificadores e endentação, que precisam ser seguidas.

Do ponto de vista prático, sugere-se ao instrutor que dê ênfase a sessões de laboratório nas quais devem ser demonstradas diversas técnicas de programação, de modo que o aluno desenvolva a necessária experiência em programação. O uso de IDEs sofisticados não é recomendado, visto que o aluno poderá desperdiçar muito tempo tentando aprender a utilizar tais ambientes de programação.

O instrutor também deve oferecer suporte suficiente à resolução das listas de exercícios de programação encontradas ao final de cada capítulo. A esse respeito, sugere-se a adoção de uma metodologia de avaliação de programas que valorize os critérios de **funcionalidade do programa** e **estilo de programação**. Por outro lado, a análise de eficiência de programas pode ser adiada para uma disciplina dedicada ao estudo de algoritmos e estruturas de dados.

A título de ilustração, descreve-se a seguir uma proposta que pode servir de guia para a tarefa de avaliação de programas. De acordo com essa proposta, considera-se que cada programa valeria de 0 a 10 pontos e que a avaliação em cada um dos dois critérios apontados acima corresponderia a, no máximo, cinco pontos. Então, são desenvolvidos sub-critérios para cada uma das categorias de avaliação, de modo a reduzir a subjetividade na avaliação dos programas.

A tabela a seguir apresenta algumas prováveis características apresentadas por programas classificados com os respectivos pontos no critério de funcionalidade do programa.

Funcionalidade do Programa	
Pontuação	**Características do Programa**
5	• Absolutamente (ou, pelo menos, convincentemente) correto e robusto.
4	• Quase correto, mas contém alguns *bugs* triviais que não comprometem o funcionamento do programa como um todo. • Não prevê todas as entradas possíveis para o programa (inclusive aquelas inválidas) e pode abortar em situações não usuais.

Funcionalidade do Programa	
Pontuação	**Características do Programa**
3	• Basicamente correto, mas contém alguns erros que comprometem seu funcionamento. • Aborta com relativa facilidade.
2	• Abordagem de solução do problema é razoável, mas contém muitos erros de implementação. • Aborta com muita facilidade.
1	• Basicamente, apenas sintaticamente correto. • O programa funciona, mas os resultados são errados.
0	• Não consegue sequer ser compilado devido a erros de sintaxe.

A tabela seguinte sugere algumas prováveis características apresentadas por programas classificados com os respectivos pontos no critério de estilo de programação.

Estilo de Programação	
Pontuação	**Características do Programa**
5	• Segue todas as recomendações básicas de estilo encontradas no livro.
4	• Utiliza algoritmos e estruturas de dados adequados. • Apresenta uma boa interface do usuário em termos de apresentação e facilidade de uso. • É bem documentado. • É tão simples quanto possível. • Os identificadores são bem escolhidos e é fácil de ler e entender.

Estilo de Programação	
Pontuação	**Características do Programa**
3	• Algoritmos e estruturas de dados não foram bem escolhidos. • A interface do usuário é rudimentar. • Documentação e clareza são descuidadas. • É mais complexo do que deveria. • Os identificadores não são representativos, mas é razoavelmente fácil de entender.
2	• Algoritmos e estruturas de dados ruins. • A interface do usuário é tão ruim que apenas o próprio programador sabe usar. • Documentação ruim ou ausente. • Difícil de ser entendido.
1	• Não segue um algoritmo claro ou as estruturas de dados são escolhidas a esmo. • A interface do usuário é tão ruim que, às vezes, o próprio programador esquece como usá-la. • Entende-se com muito sacrifício.
0	• Ilegível. • Não é um trabalho digno de um aluno de programação.

Material Complementar

Hardware e Software

Este livro precisa ser complementado com um ambiente de laboratório. Praticamente, qualquer computador com menos de dez anos de fabricação serve como hardware. A boa notícia é que o software é gratuito e facilmente encontrado na internet. As tabelas a seguir apresentam sugestões de software para as plataformas Linux e Windows.

Linux	
Software	**Comentários**
Ambiente IDE: Anjuta (Gnome) ou KDevelop (KDE)	• São bons ambientes de programação, mas são um tanto complexos para aprender a programar.
Compilador: gcc	• Instalado automaticamente em qualquer distribuição Linux. • Apesar de gratuito, é atualmente o melhor compilador de C. • Utilize a versão mais recente, pois a compatibilidade com C99 está em contínuo desenvolvimento.
Editor de programas: SciTE KWrite (KDE), vi ou Emacs	• Os editores SciTE e KWrite são bem mais fáceis de usar do que vi ou Emacs, mas estes últimos são bem mais poderosos.
make	• É um utilitário que faz parte de qualquer distribuição de Linux.
splint	• É uma ferramenta de programação que verifica arquivos-fonte escritos em C baseada no programa lint original desenvolvido para Unix. • É facilmente encontrado na internet em versões para Linux e Windows.

Windows	
Software	**Comentários**
Ambiente IDE: DevC++	• Não é um IDE para programadores sérios, mas é o mais fácil de usar que se conhece. • Utiliza uma versão capenga de gcc.

Compilador: gcc instalado com Cygwin	• Instalação não é trivial • Porta do gcc não é das melhores
Editor de programas:TextPad	• Fácil e agradável de usar, mas não possui grandes recursos. • Oferece mais suporte para Java do que para C/C++.
make	• Instalado automaticamente com o ambiente DevC++; procure-o no diretório de instalação deste ambiente.
splint	• Ver comentários na tabela anterior.

Material exposto na internet muda constantemente de lugar. Por isso, evitou-se fazer referências sobre material complementar encontrado na internet usando URLs. No site dedicado ao livro na internet encontram-se abundantes referências a sites onde se podem encontrar ferramentas de programação, exemplos de códigos-fonte, artigos e outras referências úteis para aprender e aprimorar seu conhecimento sobre programação em C.

Exemplos e Códigos-fonte

No site do livro da internet, *http://www.ulysseso.com/progc.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*. Enfim, você provavelmente não terá dificuldades para baixá-lo utilizando qualquer navegador de *web*.

Os exemplos de programas foram testados usando os compiladores Borland C++ versão 5.0.2, DevC++ versão 4.9, no sistema operacional Windows 2000, e gcc 4.1.2, na distribuição Ubuntu 6.10 do sistema operacional Linux. Se você utilizar outro sistema operacional ou compilador aderente ao padrão C99, provavelmente, não precisará fazer nenhuma alteração nos programas para colocá-los em funcionamento.

Convenções Adotadas no Livro

Convenções Tipográficas

Este livro utilize *itálico* nas seguintes situações:

- Para enfatizar a determinado ponto.
- Para representar componentes de construções da linguagem C ou de comandos de sistemas operacionais, compiladores, depuradores, etc. que devem 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 (**NB**: palavras de origem estrangeira, como array e buffer, reconhecidas pelos principais dicionários brasileiros não são representadas desta maneira).

O livro utiliza **negrito** quando:

- Conceitos são definidos.
- Palavras-chaves 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 são 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 mensagens emitidas por um programa.

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

Outras Convenções

Alterações introduzidas pelo novo padrão ISO serão identificadas no texto por (C99). É importante que o leitor dê atenção a esta indicação, pois as mudanças introduzidas pelo padrão C99 foram implementadas em poucos compiladores. Logo, uma característica particular introduzida por C99 pode não ter sido implementada em seu compilador.

Na descrição de construções sintáticas da linguagem, as palavras em itálico representam guardadores de lugares (ou, mas precisamente, símbolos metalingüísticos não-terminais). Por exemplo, no modelo sintático a seguir:

while (*expressão*)
instrução;

expressão e *instrução* representam, respectivamente, uma expressão e uma instrução válidas da linguagem C, enquanto que **while** é uma palavra da própria linguagem C (i.e., um símbolo terminal desta linguagem). A adoção desta representação esquemática, certamente, não é tão precisa quanto o uso de BNF ou grafos sintáticos, mas é mais acessível e o aluno não precisa estudar formalismos para representação sintática.

O uso de acentuação infelizmente ainda apresenta problemas de portabilidade. Por exemplo, enquanto é possível apresentar no terminal palavras acentuadas em distribuições de Linux com suporte para Unicode o mesmo não é possível em alguns outros sistemas operacionais. Assim, foi decidido que os programas exibidos como exemplos não apresentassem palavras acentuadas no meio de saída padrão. Também, apesar de o padrão C99 recomendar o uso de caracteres universais (UCN) em identificadores e *strings*, o único compilador que implementa adequadamente essa novidade é gcc em suas versões mais recentes e, mesmo assim, o faz apenas para *strings*. Portanto, como é usual na maioria das linguagens de programação, identificadores continuam sendo escritos sem acentuação ou cedilha.

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 ambigüidade. De modo análogo, quando existe impendente ambigüidade na representação gráfica de números octais e decimais, os subscrito 8 e 10 são utilizados, respectivamente.

Como é comum em livros sobre programação em C e algumas outras linguagens, todas as funções são seguidas de um par de parênteses [por exemplo, **printf()**].

Três pontos (...) num fragmento de programa representam um trecho de programa (i.e., declarações e instruções) omitidas por não serem relevantes na discussão em tela. Se

desejar usar algum desses fragmentos num programa, os três pontos devem ser convenientemente substituídos (às vezes, pode-se simplesmente removê-los).

As convenções utilizadas na escrita de identificadores são aquelas apresentadas no **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: `tNo`.
- 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`.

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 nos **Capítulos 2** e **4**. 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 seja capaz de 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, diferentemente de algumas outras linguagens, faz inequívoca distinção entre estes conceitos. Apesar disso, *declaração* é eventualmente utilizado 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 (i.e., uma coleção de bytes armazenadas em disco rígido, por exemplo), eles não aparentam ter dificuldade em

entender esse 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. *Linux não é Unix* é uma frase mencionada *ad nauseum* na internet, mas, do ponto de vista das menções apresentadas neste livro não há nenhuma diferença entre esses sistemas.

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 específicos 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, deu-se preferência ao uso de *arquivo de cabeçalho* ao invés de *cabeçalho*, como seria mais preciso.

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, ao invés do acesso direto promovido por variáveis comuns. Alguns autores e tradutores nacionais têm utilizado, com o mesmo significado, o termo *de-referenciação*, que, além de também não ser vernacular, é muito mais difícil de explicar o significado ou origem.

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 um array* ou *retorna um ponteiro para um array*. A mesma simplificação elíptica aplica-se a *strings*, que também são arrays.

Práticas Inimitáveis

O livro adota práticas que o programador deve evitar em programação real. Longe de ser mais um exemplo de *faça-o-que-digo-mas-não-faça-o-que-faço*, a adoção destas práticas possuem 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*. Em vários trechos do 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 é que estes exemplos não constituem programas completos e, sendo assim, não existe o 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, ao invés de assumir que uma dada operação sempre obtém êxito. Em muitos exemplos apresentados no texto, estes valores não são testados simplesmente para não desviar atenção daquilo que o exemplo pretende enfocar.
- *Uso de tipos inteiros primitivos*. Os únicos tipos inteiros primitivos de C portáveis são s**igned 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, é recomendável que o programador utilize os tipos inteiros de tamanho definido incorporados no arquivo de cabeçalho `<stdint.h>` (C99).

Críticas, Sugestões e Comentários

Apesar de a maior parte do material já ter sido utilizada durante anos em forma de notas de aula e, conseqüentemente, ter passado por inúmeras correções, sua expansão inevitavelmente introduziu 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 e você acredita tratar-se de um erro de programação 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/progc.htm*. Neste site, também encontra-se vasto material que complementa o livro, em particular, e sobre programação em C, em geral.

Agradecimentos

Este livro é oriundo de notas de aula continuamente refinadas durante vários semestres de ensino da disciplina Linguagem de Programação I no curso de Ciência da Computação da Universidade Federal da Paraíba. Deste 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. O professor Aluizio Araújo do Centro de Informática da UFPE leu este prefácio e apresentou inúmeras críticas e sugestões; destarte, merece reconhecimento especial.

Ulysses de Oliveira
Março de 2007