Capítulo 1

Introdução às Linguagens de Programação

Após estudar este capítulo, você deverá ser capaz de:

- Definir os seguintes conceitos:
  - ❒ Linguagem de máquina
  - ❒ Linguagem assembly
  - ❒ Linguagem de baixo nível
  - ❒ Linguagem de alto nível
  - ❒ Variável
  - ❒ Variável simbólica
  - ❒ Tradutor
  - ❒ Assembler
  - ❒ Compilador
  - ❒ Interpretador
  - ❒ Programa-fonte
  - ❒ Programa-objeto
  - ❒ Programa executável
  - ❒ Programa interativo
  - ❒ Programa de console
  - ❒ IDE
  - ❒ Editor de programas
  - ❒ Biblioteca
- Descrever as seguintes propriedades desejáveis de um programa de boa qualidade:
  - ❒ Legibilidade
  - ❒ Manutenibilidade
  - ❒ Portabilidade
  - ❒ Eficiência
  - ❒ Reusabilidade
  - ❒ Robustez
  - ❒ Usabilidade
  - ❒ Confiabilidade
- Descrever as tecnologias representadas por MinGW, GCC e CodeBlocks
- Explicar o processo de criação de um programa executável e os papéis desempenhados por compilador e linker nesse processo
- Comparar compilação e interpretação
- Explicar por que um programa interpretado leva mais tempo para ser executado do que um programa compilado equivalente
- Identificar quando dois programas são considerados funcionalmente equivalentes
- Expor o conceito de cadeia de ferramentas de desenvolvimento (*toolchain*)
- Justificar o uso de um bom estilo de programação
- Seguir recomendações para melhorar o aprendizado de programação

OBJETIVOS

1.1 Um Breve Histórico de Linguagens de Programação

PARA UM COMPUTADOR executar uma dada tarefa é necessário que ele seja informado, de uma maneira clara, *como* deve executá-la. Em outras palavras, para cada tarefa a ser executada, o computador deve ser **programado**. Infelizmente, a **linguagem nativa** ou **linguagem de máquina** de um computador (i.e., a linguagem com a qual ele é dotado durante sua fabricação) é muito limitada em dois aspectos:

- Qualquer instrução em linguagem de máquina deve ser representada por números binários formados apenas com sequências de zeros e uns.
- Computadores entendem apenas um conjunto relativamente pequeno de instruções simples como, por exemplo, *mova tal número de uma posição para outra em memória* ou *some dois números inteiros*. Cada tipo de computador (i.e., processador) possui um **conjunto de instruções** específico dessa natureza.

A memória de um computador é dividida em unidades, cada uma das quais possui um endereço único. Em programação, **variável** consiste numa unidade ou num agrupamento de duas ou mais dessas unidades contíguas. Uma variável é caracterizada por seu endereço e seu conteúdo, sendo que o endereço de uma variável constituída de várias unidades de memória corresponde ao endereço de sua primeira unidade. Variáveis servem para armazenar os dados que serão processados por um programa.

Agora, considere, por exemplo, uma operação que soma dois números inteiros armazenados em duas variáveis e guarda o resultado numa terceira variável. Uma hipotética instrução capaz de realizar essa tarefa na linguagem de máquina de um determinado computador poderia ser representada como é visto na **Figura 1–1**.

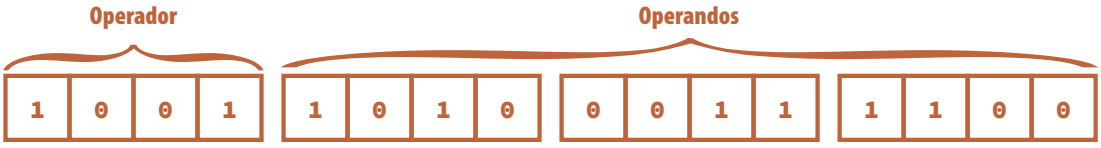

FIGURA 1–1: INSTRUÇÃO HIPOTÉTICA EM LINGUAGEM DE MÁQUINA

Nessa instrução, *operador* representa a operação que será efetuada. Ou seja, na hipotética instrução do exemplo, o número binário `1001` está associado à operação de soma. Por outro lado, os operandos representam os dados sobre os quais essa operação incide. No presente exemplo, a instrução deve somar dois valores que se encontram em memória e armazenar o resultado numa variável. Em programação em linguagem de máquina, variáveis são referenciadas por meio de seus endereços. Logo os dois primeiros operandos (`1010` e `0011`) são os endereços dos valores que serão somados e o terceiro operando (`1100`) é o endereço da variável na qual o resultado será armazenado.

A instrução ilustrada na **Figura 1–1** é hipotética, mas é plausível. Quer dizer, provavelmente, não existe nenhum processador que possua uma instrução igual a essa, mas várias instruções em linguagem de máquina real contêm um operador representado por uma sequência de bits e os operandos (variáveis) são referenciados por seus endereços, cada um dos quais também é uma sequência de bits.

Apesar de a operação ilustrada no exemplo acima ser muito comum em computadores, é possível que dois tipos diferentes de processadores a representem de modos diferentes em linguagem de máquina. Assim, a instrução em linguagem de máquina que pode ser usada para somar dois números num determinado processador pode não ser reconhecida como válida em outro processador.

Como você já deve ter desconfiado, programas escritos em linguagem de máquina são difíceis de escrever, ler, modificar e portar de um tipo de tipo de computador para outro. Programação em linguagem de máquina é

considerada **programação de baixo nível** porque está muito próxima à máquina, mas bem distante da linguagem humana.

Uma sensível evolução em relação à linguagem de máquina foi o surgimento da **linguagem assembly**, no final da década de 1940, que introduziu os seguintes melhoramentos:

- Uso de **palavras mnemônicas** para representar operações escritas como códigos binários em linguagem de máquina. *Mnemônico* refere-se a artifícios que auxiliam a memorização. Em assembly, palavras mnemônicas deveriam lembrar as operações que representam, mas elas não são tão mnemônicas para muitos programadores que não dominam a língua inglesa. Por exemplo, o operador `1001` usado para adicionar números do exemplo acima poderia ser identificado como `ADD` em assembly.
- Uso de **variáveis simbólicas** em vez de endereços para representar espaços na memória do computador. Por exemplo, o operando (variável) do exemplo acima referenciado pelo endereço `1100` em linguagem de máquina poderia ser identificado em assembly por, digamos, `X`.

Utilizando assembly, a instrução hipotética do exemplo anterior (v. **Figura 1–1**) poderia ser escrita como na **Figura 1–2**.

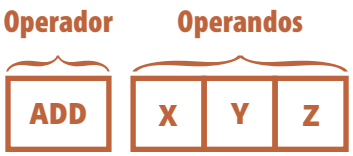

Figura 1–2: Instrução Hipotética em Linguagem Assembly

Apesar de assembly constituir um melhoramento considerável para o programador em relação à linguagem de máquina, programas em assembly ainda são difíceis de escrever, ler, modificar e portar. Assim, programação em assembly também é considerada de baixo nível.

Até meados da década de 1950, todos os programas de computador eram de baixo nível (i.e., escritos em linguagem de máquina ou assembly). Facilidades para o programador bem maiores do que aquelas introduzidas por assembly foram incorporadas pelas **linguagens de alto nível**, que permitem que programas sejam escritos usando uma linguagem *mais* próxima da linguagem humana. Essas linguagens permitem que o programador escreva programas sem ter que preocupar-se com detalhes da máquina que irá executá-los. Hoje em dia, a maioria dos programas é escrita em linguagens de alto nível, mas muitos programadores ainda usam assembly por questões de eficiência (v. **Seção 1.3**).

1.2 Tradutores

Todo programa escrito numa linguagem de programação que não seja a linguagem de máquina de um computador requer uma tradução para essa linguagem antes que ele possa ser executado nesse computador. **Tradutor** é um programa de computador que aceita como entrada um programa escrito em assembly ou numa linguagem de alto nível e produz como saída um programa escrito em linguagem de máquina. Há três tipos de tradutores que serão examinados a seguir.

1.2.1 Assembler

Apesar de ser uma linguagem de baixo nível, assembly é estranha a qualquer computador, pois ele só entende sua própria linguagem nativa. Assim programas escritos em assembly requerem um programa especial, chamado **assembler** (ou **montador**), que traduz programas escritos em assembly em programas escritos em linguagem de máquina.

Conforme foi visto antes, a linguagem assembly permite que o programador escreva programas usando palavras mnemônicas e nomes de variáveis, em vez de sequências de bits de uma linguagem de máquina. Cada palavra mnemônica na linguagem assembly de um determinado processador corresponde exatamente a uma operação na linguagem de máquina desse processador e vice-versa. Portanto, para cada instrução escrita em assembly, existe uma única instrução na linguagem de máquina correspondente. Mais precisamente, a única decisão que um tradutor assembler precisa tomar, quando for necessário, diz respeito aos operandos de uma instrução (p. ex., onde armazená-los). Mas, esse não é o caso com relação a operadores. Por isso, o assembler é o tipo de tradutor mais simples que existe.

1.2.2 Compiladores

Linguagens de alto nível requerem tradutores mais complexos, pois, frequentemente, uma única instrução em linguagem de alto nível pode corresponder a várias instruções em linguagem de máquina. Existem dois tipos de tradutores para linguagens de alto nível: compiladores e interpretadores.

Compilador é um programa que traduz um programa escrito numa linguagem de alto nível (**programa-fonte**) num programa escrito numa linguagem de máquina (**programa-objeto**). Um compilador depende não apenas da linguagem-fonte para a qual ele foi especificado, como também do processador em cuja linguagem de máquina os programas-fonte são traduzidos. Assim, é comum fazer-se referência a um compilador da linguagem X (p. ex., C) para o processador Y (p. ex., Intel Pentium).

Usualmente, um programa escrito numa linguagem de alto nível possui partes que são compiladas separadamente. Para reunir as partes (arquivos) compiladas de um programa de modo a constituir um **programa executável**, utiliza-se um programa chamado **linker** (ou **link editor** ou **editor de ligações**). Isto é, linker é um programa que recebe como entrada um conjunto de arquivos compilados, faz as devidas ligações entre eles e produz um programa pronto para ser executado.

Pode-se fazer uma analogia entre um compilador e um tradutor humano contratado para ler um texto numa língua desconhecida (texto-fonte) e produzir, por escrito, sua tradução numa língua conhecida (texto-objeto). Uma vez que o texto-objeto tenha sido concluído e entregue ao interessado, o tradutor pode ser dispensado, porque seu serviço não é mais necessário. O mesmo ocorre com um programa executável; i.e., após a conclusão dos processos de compilação e ligação, ele torna-se independente (**programa standalone**) e pode ser executado sem precisar contar com um compilador ou linker.

1.2.3 Interpretadores

Diferentemente de um compilador, um **interpretador** não produz programas-objeto. Em vez disso, ele simula um computador cuja linguagem de máquina seria aquela do programa-fonte que está sendo traduzido. Assim, cada instrução do programa-fonte é traduzida exatamente antes de ser executada e de acordo com o fluxo de execução do programa. Isso significa que instruções que são executadas mais de uma vez, também são traduzidas o mesmo número de vezes antes de serem executadas; instruções que não são executadas não são traduzidas. Em contraste, um compilador traduz cada instrução de um programa-fonte exatamente uma vez.

Outra diferença importante entre compiladores e interpretadores é que, uma vez compilado, um programa torna-se independente do compilador, enquanto um programa interpretado depende sempre de um interpretador para ser executado. Além disso, programas compilados são executados mais rapidamente do que programas equivalentes interpretados (você seria capaz de explicar por quê?).

Pode-se fazer uma analogia entre um interpretador e um tradutor humano contratado para ler um texto numa língua desconhecida para o contratante e produzir oralmente sua tradução numa língua conhecida pelo mesmo

interessado. Nesse caso, o tradutor precisará ser convocado novamente sempre que outra tradução do mesmo texto for necessária.

Pelo que foi exposto, parece que compiladores são sempre mais vantajosos que interpretadores. Mas, interpretadores apresentam duas vantagens em relação a compiladores:

- Durante a fase de desenvolvimento de um programa, alterações realizadas no programa são testadas mais rapidamente usando um interpretador do que usando um compilador. Isso ocorre porque, no caso de compilação, o programador tem que esperar que o programa executável seja reconstruído, o que não ocorre no caso de uso de interpretação. Essa diferença, entretanto, só é relevante no caso de grandes projetos de programação.
- No corrente contexto, **plataforma** é uma combinação de processador com sistema operacional. Por exemplo, Intel em conjunto com Windows constituem uma plataforma. Assim, um programa interpretado pode ser distribuído em forma de programa-fonte, desde que haja disponibilidade de interpretador para a plataforma para a qual o programa é destinado. Logo, um programa interpretado pode ser relativamente mais portável que um programa compilado.

Essa última vantagem dos interpretadores é um tanto quanto sutil. Afinal, você pode raciocinar do seguinte modo: se houver um compilador disponível numa determinada plataforma, também é possível distribuir o programa-fonte para um usuário dessa plataforma. Mas esse raciocínio é equivocado porque compilar e efetuar as devidas ligações de um programa-fonte requerem conhecimento técnico especializado, enquanto traduzir e executar um programa interpretado não requer praticamente nenhum conhecimento específico por parte do usuário. Por exemplo, muitas páginas da web incorporam programas-fontes (p. ex., *applets* Java) que são interpretados no computador do usuário sem que ele sequer tome conhecimento desse fato.

1.3 Qualidades de um Bom Programa

Linguagens de alto nível podem ser comparadas com linguagens de baixo nível levando-se em consideração as seguintes propriedades desejáveis de um programa:

- **Legibilidade**. Um programa que tem boa legibilidade é um programa fácil de ler e entender. O uso de uma linguagem de alto nível na escrita de um programa não garante, em princípio, que ele terá boa legibilidade. Porém, em virtude das características das linguagens de baixo nível, é praticamente impossível, mesmo para os melhores programadores, a escrita de programas legíveis com o uso de tais linguagens.
- **Manutenibilidade**. A facilidade de modificação (i.e., manutenção) que um programa apresenta é denominada *manutenibilidade*. Evidentemente, essa propriedade está intimamente relacionada com a legibilidade, pois quanto mais fácil de ser compreendido um programa é, mais fácil será modificá-lo. Assim um programa com má legibilidade provavelmente também terá má manutenibilidade. Entretanto, um programa pode apresentar boa legibilidade e, mesmo assim, não ser fácil de modificar. Novamente, o uso de uma linguagem de alto nível não garante que um programa tenha essa propriedade satisfeita, mas um programa escrito numa linguagem de baixo nível dificilmente terá boa manutenibilidade.
- **Portabilidade.** Um programa portável é aquele que pode ser transportado de uma plataforma para outra com pouca ou, idealmente, nenhuma alteração do programa-fonte. Um programa escrito numa linguagem de baixo nível é específico para um tipo de processador; i.e., se o programador desejar executar esse programa em outro tipo de processador, ele terá que ser substancialmente reescrito. Por outro lado, programas escritos para uma dada plataforma usando uma linguagem de alto nível podem ser transportados para outra plataforma com relativa facilidade. O grau de portabilidade de um

programa depende da habilidade do programador, bem como das plataformas em questão e da natureza do programa. Por exemplo, tipicamente, um programa que apresenta interface gráfica é bem mais difícil de portar de uma plataforma para outra do que um programa cuja interface é baseada em console.

- **Eficiência**. A eficiência de um programa é medida pelo espaço que ele ocupa em memória e pela rapidez com que é executado. Um programa eficiente ocupa apenas o espaço estritamente necessário e tem uma execução relativamente rápida. Em linguagem de máquina, diferentes sequências de instruções podem resultar em programas que são funcionalmente equivalentes (i.e., que produzem os mesmos resultados), mas algumas sequências de instruções funcionalmente equivalentes podem ser mais eficientes do que outras. Escrevendo um programa em linguagem de baixo nível, um bom programador pode decidir qual sequência de instruções é a mais eficiente. Por outro lado, escrevendo um programa numa linguagem de alto nível, o programador tem pouco controle sobre como um compilador ou interpretador traduz o programa para linguagem de máquina. Infelizmente, a realidade é que mesmo compiladores ou interpretadores sofisticados podem produzir programas executáveis ineficientes.
- **Reusabilidade** é a facilidade com que um programa permite que suas partes sejam reutilizadas em outros programas. Linguagens modernas de alto nível, notadamente aquelas orientadas a objetos, favorecem essa propriedade, mas, novamente, é responsabilidade do programador garantir que seus programas satisfazem adequadamente essa propriedade. Reusabilidade é uma propriedade de crucial importância no ensino de programação que merece atenção especial neste livro. A **Seção 7.2** discutirá este tópico em maiores detalhes e diversas outras seções mostram como ela é aplicada na prática.

Outras qualidades de um bom programa que merecem ser mencionadas são:

- **Robustez**. Um programa robusto é capaz de lidar adequadamente com situações que não são consideradas normais (**condições de exceção**). Como exemplos de condições comuns de exceções podem ser citados: dados inconvenientes introduzidos pelo usuário, impossibilidade de abertura de um arquivo, ocorrência de erro na leitura de um arquivo e esgotamento de memória. Dotar um programa de completa robustez requer um estudo aprofundado de tratamento de exceção que está além do escopo deste livro. Mesmo assim, tenta-se dotar os programas apresentados neste livro de robustez aceitável para o nível de aprendizagem pretendido.
- **Usabilidade** é a facilidade de uso apresentada por um programa executável. Diferentemente das demais propriedades, essa propriedade diz respeito ao usuário final do programa. Devido à simplicidade dos programas considerados neste livro, o melhor que eles podem oferecer são prompts (v. **Seção 3.14.3**) que facilitem o entendimento daquilo que o programa espera que o usuário introduza e a exibição de resultados que facilitem a interpretação do usuário.
- **Confiabilidade**. Idealmente, os resultados de um programa devem estar sempre corretos e, em virtude da simplicidade dos programas discutidos neste livro, esse deve ser o caso a ser levado em consideração. Mas, em determinadas circunstâncias, um programa mais complexo comete erros e, mesmo assim, é aceitável. Por exemplo, muitos programas comercialmente disponíveis são repletos de erros de programação (**bugs**), mas, apesar disso, podem alcançar milhões de usuários (portanto, são aceitáveis).

Resumindo, todo bom programador deve saber construir programas que atendam adequadamente as propriedades descritas acima. Um dos objetivos deste livro é ajudar o programador a adquirir um **estilo de programação** em linguagem C que o capacite a escrever programas que apresentem as boas qualidades descritas nesta seção, com exceção de eficiência. A justificativa para a não inclusão de eficiência como tópico prioritário num livro introdutório deve-se ao conhecimento requerido para sua completa compreensão. Ou seja, para um programador de linguagem de alto nível ser capaz de criar programas eficientes, ele deve possuir muito conhecimento sobre como um programa é compilado, ligado, carregado em memória e, finalmente, executado.

1.4 Programas Interativos Baseados em Console

Um dos principais objetivos deste livro é ensinar a construir programas interativos baseados em console (ou terminal). Um **programa interativo** é aquele que dialoga com o usuário. Ou seja, ele permite que o usuário introduza dados e lhe apresenta o resultado do processamento desses dados. Nem todo programa é interativo. Por exemplo, existem programas, denominados controladores (*device drivers*), que se comunicam com sistemas operacionais (e não com usuários). A maioria dos vírus de computador também não se comunica com usuários (infelizmente...).

Programas baseados em **console** (ou **terminal**) usam apenas **teclado** como dispositivo de entrada (leitura) de dados e a **tela** do computador (ou **monitor de vídeo**) como dispositivo de saída (escrita) de dados. Programas dessa natureza são baseados apenas em texto e não usam entrada via mouse ou interfaces gráficas, como aquelas características de sistemas Microsoft Windows ou Gnome e KDE (Linux).

Os programas enfocados neste livro podem causar alguma frustração ao leitor, visto que a maioria dos programas em uso nos dias atuais é de natureza gráfica. Contudo, programação usando interfaces gráficas envolve conceitos e técnicas que, além de acrescentarem muita complexidade, pouco têm a ver com programação em si. Estar envolvido simultaneamente com esses conceitos enquanto se aprende a programar é um fator de distração do objetivo principal. Além disso, programas baseados em texto ainda são usados em profusão.

1.5 Ambientes Integrados de Desenvolvimento

Um **ambiente integrado de desenvolvimento** (ou **IDE**) é um programa que coordena a execução de vários outros programas dedicados ao desenvolvimento de programas. Tipicamente, um IDE está associado a pelo menos os seguintes programas de desenvolvimento[1]:

- **Editor de programas**. Um editor de programas é semelhante a um editor de texto comum, mas possui a vantagem adicional de, pelo menos, conhecer a sintaxe de uma linguagem de programação. A vantagem de usar um editor de programas em vez de um simples editor de texto em desenvolvimento é que um editor de programas não apenas é capaz de facilitar bastante a edição de programas como também de indicar possíveis erros de sintaxe antes mesmo de o programa ser compilado. Todo IDE decente possui um editor de programas associado (v. adiante).
- **Compilador**. Esse programa evidentemente é imprescindível, pois é o responsável pela tradução de programas-fonte em programas-objeto. Nesse caso, a tarefa de um IDE é executar o compilador sem que seja necessário abandoná-lo. Qualquer IDE possui a capacidade de integrar esse componente.
- **Editor de Ligações** (**Linker**). Esse programa também é imprescindível num IDE, pois é ele quem realmente cria os programas executáveis. Aqui, novamente, a tarefa do IDE é permitir a execução do linker sem que seja necessário abandoná-lo. Qualquer IDE é capaz de integrar esse componente.
- **Carregador** (**Loader**). Antes que um programa seja executado, ele é carregado em memória e, então, preparado para execução. Essas tarefas são efetuadas por um programa denominado **loader**, que faz parte do sistema operacional utilizado como hospedeiro do programa. Logo após a conclusão dessa etapa preparatória, o sistema operacional inicia a execução do programa. Um IDE capaz de executar um *loader* permite que o programador execute seus próprios programas sem ter que abandonar o IDE. A maioria dos IDEs inclui essa capacidade.
- **Depurador** (**Debugger**). Esse programa ajuda o programador a encontrar e corrigir erros de programação e não é imprescindível como os demais, mas é bem desejável.

[1] IDE é derivado de *Integrated Development Environment*, em inglês. IDEs que usam interpretadores não são abordados neste livro.

Seja paciente quando lidar com ambientes de desenvolvimento, pois eles não são tão tolerantes a erros do usuário como ocorre com programas aplicativos modernos.

A maior parte do tempo de um programador é dedicado a edição de programas. Qualquer editor de texto (não confunda com *processador* de texto) pode ser utilizado na digitação de um programa, mas o mais indicado para essa tarefa é um editor de programas, que é um editor de texto que *entende* pelo menos uma linguagem de programação e oferece, no mínimo, as seguintes vantagens:

- **Coloração de sintaxe**. Essa característica facilita a identificação visual dos diversos componentes de um programa. Alguns editores de programas já vêm com uma configuração de coloração padrão que agrada a maioria dos programadores. Outros editores de programas, por outro lado, vêm com uma configuração de coloração exagerada e de gosto bastante duvidoso. Em qualquer caso, a maioria dos editores de programas permite que o usuário altere a configuração do padrão de cores. Utilize essa facilidade com coerência e parcimônia e não transforme a visualização de seu programa num maracatu.
- **Endentação automática**. Endentação é o espaçamento horizontal que algumas instruções de um programa apresentam com relação a outras. As vantagens obtidas com o uso de endentação adequada são enfatizadas em vários pontos deste livro. A facilidade de endentação automática permite que o programador não tenha que fazer endentação manualmente, economizando, assim, tempo e esforço físico despendidos em digitação. Como no caso de coloração de sintaxe, os espaços de endentação automática também podem ser configurados. Três ou quatro espaços em branco são considerados a configuração ideal de endentação.

A maioria dos editores de programas oferecem outras características úteis, tais como complementação de código e emparelhamento de parênteses, chaves e colchetes, mas aquelas descritas acima são suficientes para começar a programar.

1.6 Instalação de um Ambiente de Trabalho

Como se afirma insistentemente neste livro, não é possível aprender a programar sem praticar programação. Se você adquiriu este livro com a expectativa de que irá aprender a programar apenas lendo-o, desperdiçou seu investimento e, se insistir com essa visão, também desperdiçará seu tempo. Logo comece a praticar a partir deste instante instalando um ambiente de trabalho que servirá como seu laboratório de aprendizado de programação.

O ambiente de trabalho com o qual lida este livro diz respeito a sistemas operacionais da família Microsoft Windows, visto que, provavelmente, esse sistema é o mais utilizado entre aprendizes de programação. Leitores que já utilizam algum sistema da família Unix (notadamente aqueles que usam Linux e Mac OS X) encontrarão informações sobre instalação e uso de outros ambientes de trabalho no site dedicado ao livro na internet.

A criação de um ambiente adequado de aprendizagem levará algum tempo, mas, em compensação, você só precisará fazer isso uma vez e, acredite, será bem recompensado. Siga as recomendações apresentadas a seguir na ordem que se encontram.

1.6.1 Organização

Antes de instalar um ambiente de desenvolvimento, se estiver usando Windows, crie um diretório (pasta) de trabalho, idealmente na raiz do disco C, com um nome curto e sem espaços em branco (p. ex., `Programas` é um bom nome). Para facilitar ainda mais, crie um atalho para essa pasta e coloque-o num local de fácil acesso (p. ex., na área de trabalho). Não é necessário armazenar os arquivos executáveis correspondentes aos exercícios, pois eles podem ser facilmente obtidos novamente, como você logo descobrirá. Uma boa ideia é fazer download dos exemplos deste livro, que se encontram no site ***www.ulysseso.com/ip***, e armazená-los nesse diretório.

1.6.2 Ferramentas de Desenvolvimento MinGW

Conforme foi discutido na **Seção 1.2**, todo programa escrito numa linguagem de alto nível requer um compilador ou interpretador. Para a linguagem C, que será utilizada para ensino de programação no presente livro, usa-se apenas compilador.

Todo compilador de C deve vir acompanhado de uma **biblioteca padrão** de componentes que o programador usa para construir seus programas. Essa biblioteca dota a linguagem de um caráter mais pragmático, facilitando a escrita de programas.

De acordo com o que foi visto na **Seção 1.2.2**, quando um programa usa partes que são compiladas separadamente, como é o caso dos componentes da biblioteca mencionada, é necessário um linker para fazer as devidas ligações entre o programa e os componentes que ele usa.

Diante do exposto, pode-se concluir que um compilador deve vir acompanhado de uma biblioteca e de um linker. De fato, normalmente, um fabricante de compilador raramente distribui apenas esses três componentes. Isto é, mais comumente, os fabricantes de compiladores de C distribuem outros componentes que auxiliam o desenvolvimento de programas. Excluindo-se a biblioteca, os demais componentes (que são programas) são denominados **ferramentas de desenvolvimento** (*toolchain*, em inglês). Entretanto, é largamente difundido o uso da palavra *compilador* como significado genérico para esse conjunto de ferramentas de desenvolvimento.

MinGW consiste numa coleção de ferramentas de desenvolvimento para Microsoft Windows, dentre as quais interessam sobretudo para o programador de C: compilador (**GCC**), linker (g++) e depurador (**GDB**). O pacote MinGW oferece as seguintes vantagens para o programador de C:

- O compilador GCC produz programas executáveis bastante eficientes.
- O depurador de GDB é uma excelente ferramenta de aprendizagem e depuração e pode ser usado com programas escritos em diversas linguagens.
- As ferramentas funcionam de modo equivalente nos sistemas Windows, Unix, Linux e Mac OS X.
- Ocorrem atualizações relativamente frequentes do pacote.
- Possui uma enorme rede de usuários, o que facilita a obtenção de ajuda via internet.
- É gratuito.

Se você for usar o ambiente de desenvolvimento CodeBlocks (o que é mais recomendável para aprender a programar), passe adiante para a **Seção 1.6.3**. Se você for familiarizado com outro ambiente de desenvolvimento (p. ex., DevC++) e preferir utilizá-lo ou se desejar utilizar compilação e ligação em linha de comando, busque informações para instalação do pacote mais recente de MinGW no site do livro. Essa última opção não é recomendada para quem realmente é iniciante em programação.

1.6.3 Ambiente de Desenvolvimento CodeBlocks

O ambiente de desenvolvimento (IDE) recomendado neste livro é CodeBlocks e as razões para essa recomendação são as seguintes:

- Esse IDE é facilmente encontrado na internet.
- Ele possui um ótimo editor de programas.
- Ele é capaz de incorporar simultaneamente vários compiladores e linguagens.
- Pode ser utilizado em desenvolvimento profissional (i.e., ele não serve apenas para aprendizagem de programação, como, por exemplo, o IDE DevC++).
- É muito fácil de usar.
- É gratuito.

Enfim, CodeBlocks é um excelente ambiente de desenvolvimento (IDE), desde que o iniciante não se assuste com ele. Esse IDE constitui a opção mais recomendável para usuário de Windows e, para instalá-lo, siga os seguintes passos:

1. Visite o site dedicado ao CodeBlocks, navegue até a página de downloads e obtenha o respectivo arquivo binário.
2. Se você usa Windows e ainda não instalou o pacote de desenvolvimento MinGW (v. **Seção 1.6.2**), faça download do arquivo que inclui *mingw* em seu nome. Se você possui MinGW instalado em seu computador, é melhor removê-lo antes de instalar CodeBlocks.
3. Para instalar o IDE CodeBlocks no Windows, aplique um clique duplo sobre o ícone do arquivo obtido no passo anterior e siga todas as sugestões de instalação oferecidas.

Após instalação, você deverá ser capaz de encontrar e executar esse programa no menu *Iniciar* do sistema operacional Windows. Mas, para facilitar ainda mais o acesso ao programa, crie um ou mais atalhos dele e copie-os para locais de fácil acesso (p. ex., a área de trabalho e a pasta sugerida na **Seção 1.6.1**).

1.6.4 Biblioteca LeituraFacil

A **biblioteca LeituraFacil** foi desenvolvida para o ensino de disciplinas introdutórias de programação usando a linguagem C. Ela evita as dificuldades e frustrações com que se depara um iniciante em programação em C, tornando leitura de dados via teclado tão simples quanto ela aparece numa linguagem algorítmica, como aquela que será apresentada no **Capítulo 2**.

Essa biblioteca é composta por dois arquivos:

- **Arquivo-objeto**. Esse arquivo contém o código compilado da biblioteca para uso com MinGW (Windows)[2]. O nome desse arquivo é `libleitura.a`.
- **Arquivo de cabeçalho**. Esse é um arquivo de texto, denominado `leitura.h`, que possui o mesmo conteúdo para qualquer compilador e sistema operacional.

Para fazer download da biblioteca LeituraFacil, visite o site dedicado ao livro na internet (***www.ulysseso.com/ip***) e siga o respectivo link.

Para utilizar a biblioteca LeituraFacil, é necessário armazenar o arquivo de cabeçalho e o arquivo-objeto de modo que o compilador e o linker possam encontrá-los. O procedimento para instalação e uso da referido biblioteca para ferramentas de desenvolvimento MinGW deve acompanhar os seguintes passos:

1. Copie o arquivo `libleitura.a` para a pasta `...\MinGW32\lib`, sendo que os três pontos representam o caminho até a pasta de instalação de MinGW.
2. Copie o arquivo `leitura.h` para a pasta `...\MinGW32\include`, sendo que os três pontos representam o caminho até a pasta de instalação de MinGW.

Para descobrir onde se encontra a pasta MinGW, use a ferramenta de busca do Windows.

Importante: Depois de instalar os arquivos da biblioteca LeituraFacil em suas devidas pastas, é necessário configurar o IDE CodeBlocks para que eles possam ser usados, conforme será mostrado na **Seção 1.7.2**.

1.6.5 Facilitando a Execução de Programas no Windows

Os programas que você irá criar como parte do treinamento proposto neste livro devem ser executados em linha de comando (console). Isso significa que você precisará abrir uma janela do DOS sempre que for executar um

[2] No site do livro (*www.ulysseso.com/ip*), existem outras versões desse arquivo que dependem do compilador e do sistema operacional utilizados. Consulte também o **Apêndice B**.

programa sob supervisão de um sistema operacional da família Windows. Ambientes de desenvolvimento, como CodeBlocks, permitem que o programador execute seus programas sem precisar abandoná-los. Mas, agindo sempre assim, você não entenderá completamente o processo de desenvolvimento e execução de programas e poderá adquirir um mau hábito.

Frequentemente, programadores que são usuários do sistema Windows precisam abrir janelas do DOS, o que não é uma tarefa das mais fáceis em versões antigas desse sistema (p. ex., Windows XP). Para facilitar o trabalho daqueles que usam esses sistemas antigos, existe uma extensão de interface denominada *Open command window here* ou *Abrir janela de comando aqui* (dependendo da versão de Windows à qual essa extensão se destina). Esse pequeno programa da Microsoft permite que se abra uma janela de console em qualquer pasta de maneira bem simples. Se você utiliza Windows Vista, Windows 7 ou Windows 8, não precisa instalar esse programa porque ele já faz parte das interfaces desses sistemas. Caso contrário, é preciso fazer download do programa e instalá-lo.

Uma vez que o referido programa esteja instalado, para abrir uma janela do DOS em qualquer pasta, basta clicar sobre a pasta com o botão direito do mouse e escolher, no menu que aparece, a opção *Open command window here* ou *Abrir janela de comando aqui*, como mostra a **Figura 1–3**. Nos sistemas Windows Vista, Windows 7 e Windows 8, o clique do mouse deve ser acompanhado do pressionamento da tecla `[SHIFT]`. Nesses sistemas, o comando em discussão é identificado como *Abrir janela de comando aqui*.

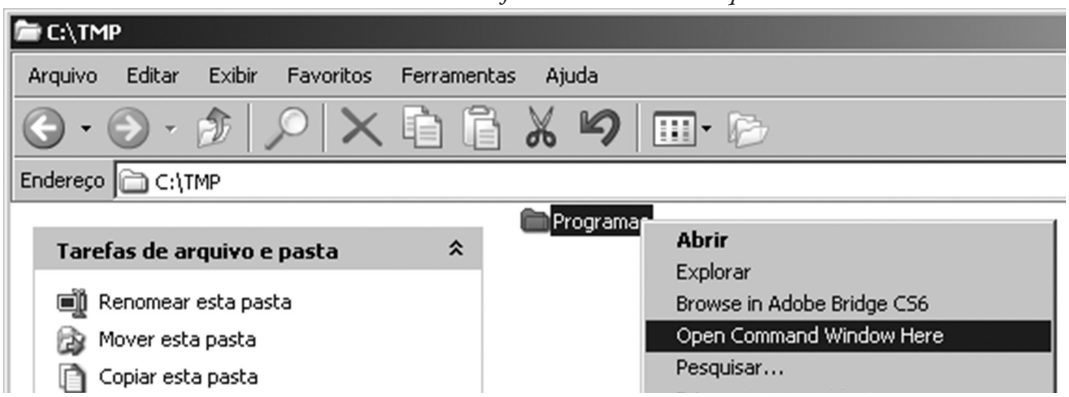

Figura 1–3: Usando a Extensão Open Command Window Here

1.7 Utilizando CodeBlocks

1.7.1 O Ambiente CodeBlocks

Quando você executa CodeBlocks[3], obtém uma janela semelhante àquela mostrada na **Figura 1–4**.

Essa janela contém três painéis:

[1] Painel da esquerda. Esse painel, que não aparece na figura, é utilizado em gerenciamento de projetos de programas mais complexos do que aqueles que serão desenvolvidos neste livro. Enquanto aprende a programar, o melhor a fazer é fechar esse painel, de modo a aumentar as dimensões dos demais painéis. Para fechá-lo, clique no botão no topo do painel ou utilize o menu View e desmarque a opção Manager.

[2] Painel da direita. Esse é o maior painel e é usado pelo editor de programas. Ele deve ser o principal foco de atenção enquanto você estiver editando um programa.

[3] Painel inferior. Nesse painel, são apresentadas mensagens geradas pelas ferramentas de desenvolvimento usadas pelo IDE CodeBlocks. Esse painel possui cinco abas, denominadas *CodeBlocks*, *Search Results* etc. A aba denominada *Build messages* é a mais importante de todas, pois é nela que são apresentadas

[3] Aqui, assume-se o uso de CodeBlocks versão 10.05 e MingGW-GCC versão 4.6.1.

mensagens de erro e advertência emitidas pelo compilador ou linker durante o processo de construção de um programa (v. **Seção 3.18**).

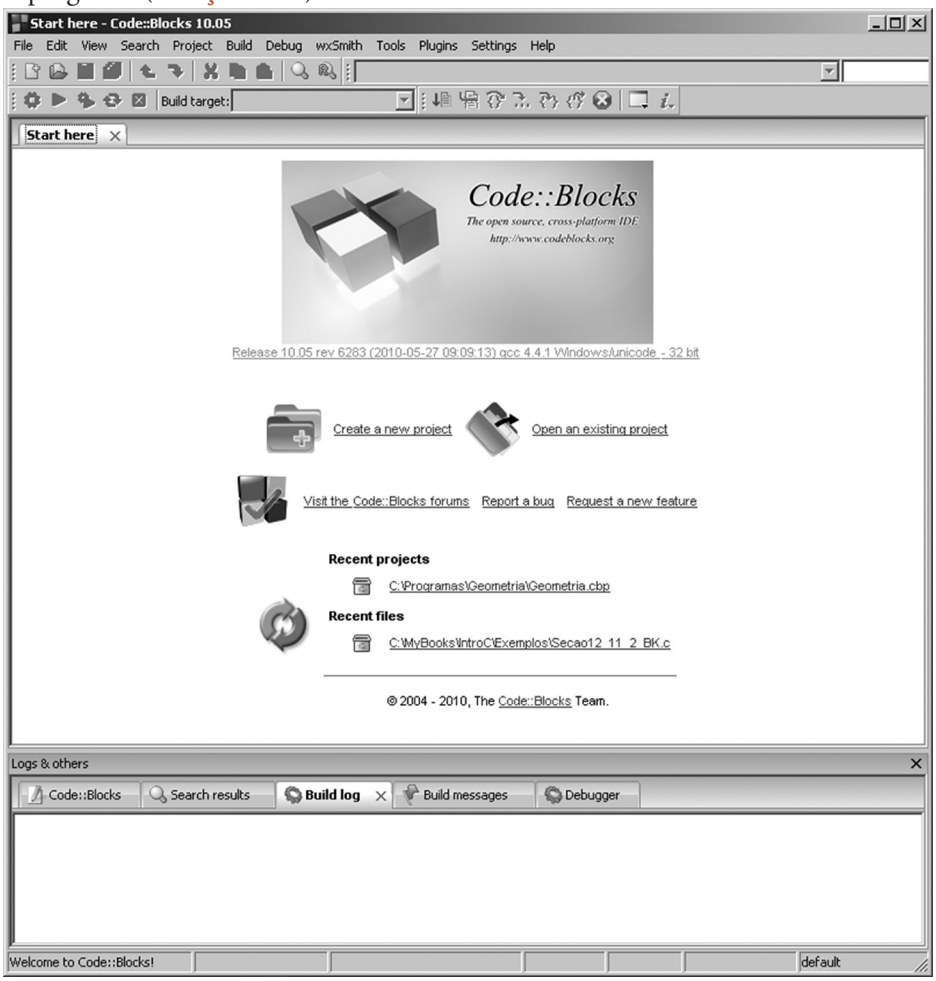

FIGURA 1–4: JANELA PRINCIPAL DO IDE CODEBLOCKS

Não se deixe assustar com a aparente complexidade do IDE CodeBlocks. Esse programa é realmente poderoso e, por isso, apresenta certa complexidade, mas ela não irá atrapalhá-lo enquanto aprende a programar. Outras características interessantes do editor do CodeBlocks que facilitam o trabalho do programador serão apresentadas oportunamente no decorrer do texto.

1.7.2 Configurando CodeBlocks com GCC (MinGW) e LeituraFacil

Antes de construir qualquer programa, você precisará configurar algumas poucas opções de compilação e ligação. Para tal, siga os seguintes passos:

1. Clique no menu *Settings* e escolha a opção *Compiler and debugger...*, como mostra a **Figura 1–5**.
2. A caixa de seleção no topo da janela que surge em seguida deve conter *GNU GCC compiler* e o botão abaixo, denominado *Set as default*, deve estar esmaecido, indicando que GCC é o compilador padrão que será utilizado. Se esse for o caso, passe para o passo a seguir. Em caso contrário, provavelmente, as ferramentas MinGW (que incluem o compilador GCC) não foram corretamente instaladas e sua melhor chance antes de instalar tudo novamente é tentar encontrar o diretório dessa instalação. Para tal, clique na aba *Toolchain executables* e, em seguida, clique no botão contendo três pontos. Então, procure

o diretório em que você instalou MinGW. Se não conseguir localizar o diretório, volte à **Seção 1.6.2** ou **1.6.3**, revise o processo de instalação de MinGW e retorne a este ponto apenas quando CodeBlocks conseguir localizar a referida instalação.

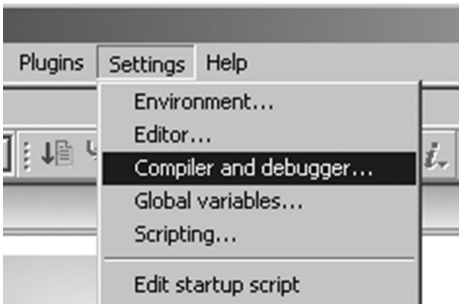

FIGURA 1–5: CONFIGURANDO CODEBLOCKS 1

3. Na aba *Compiler flags*, selecione a opção que começa com *Enable all compiler warnings*. Essa opção faz com que o compilador apresente todas as mensagens de advertência possíveis e protege o programador contra surpresas desagradáveis quando o programa for executado (v.**Seção 3.18**). Nenhuma outra opção deve ser selecionada, como mostra a **Figura 1–6**.

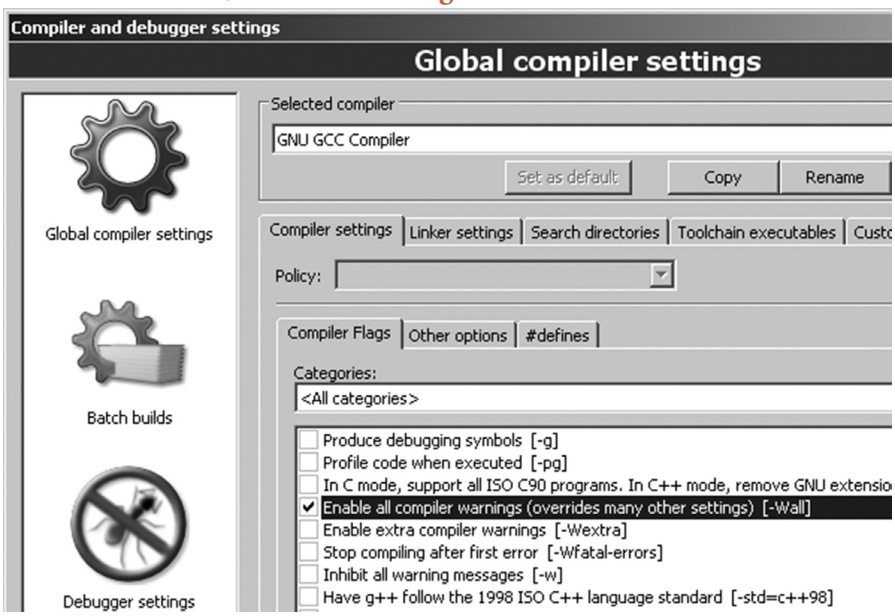

FIGURA 1–6: CONFIGURANDO CODEBLOCKS 2

4. Clique na aba *Other options* e digite no painel de edição:

```
-std=c99 -pedantic
```

(exatamente assim e não esqueça o traço antes de **`std`** e **`pedantic`**). Essas opções informam o compilador que ele deve usar o padrão ISO mais recente da linguagem C para o qual o compilador GCC oferece suporte. A **Figura 1–7** mostra o resultado desse passo.

5. Clique na aba *Linker settings* (acima da aba do passo anterior) e digite o seguinte no painel de edição *Other link options*: **`-lleitura`** (exatamente assim e não esqueça o traço no início). Essa opção permite ao linker fazer as devidas ligações entre seus programas e a biblioteca LEITURAFACIL descrita na **Seção 1.6.4**. Além disso, alguns linkers antigos que acompanham GCC precisam ser informados sobre o uso do módulo de biblioteca responsável por operações com números reais, tais como raiz quadrada, seno,

logaritmo etc. Portanto, para evitar surpresas desagradáveis, inclua também a seguinte opção: `-lm`. Ao final desse passo, você deverá obter a janela de configuração mostrada na **Figura 1–8**.

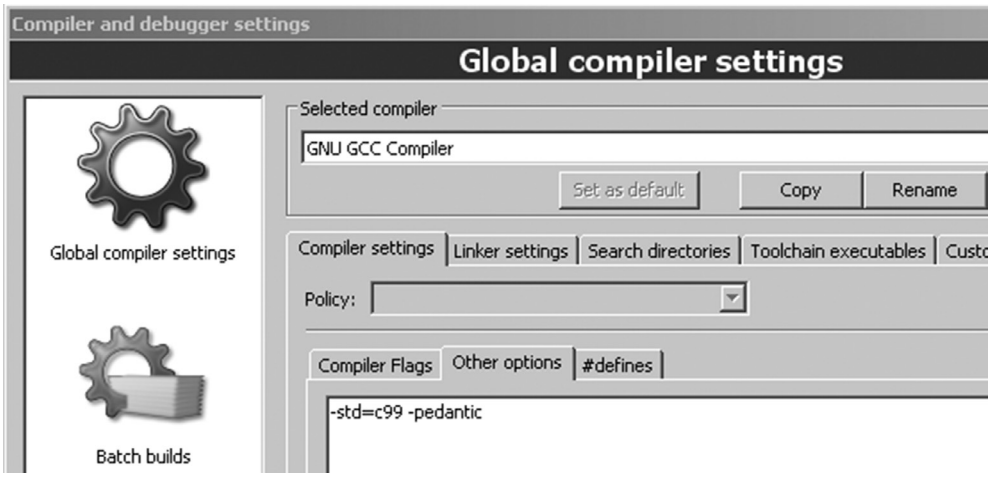

FIGURA 1–7: CONFIGURANDO CODEBLOCKS 3

FIGURA 1–8: CONFIGURANDO CODEBLOCKS 4

Provavelmente, você não precisará alterar mais nenhuma outra opção de configuração e, se você estiver utilizando CodeBlocks num computador pessoal, não precisará mais repetir essas recomendações de configuração.

1.7.3 Criando e Editando um Programa-fonte com CodeBlocks

Para criar um programa-fonte simples usando CodeBlocks, siga os seguintes passos:

1. No menu *File*, escolha o submenu *New* e, em seguida, a opção *File...*
2. Na caixa de diálogo que aparece em seguida, clique sobre o ícone denominado *C/C++ source*, como mostra a **Figura 1–9**.
3. Clique sobre o botão *Go* na janela apresentada na **Figura 1–9** e você será apresentado à janela visualizada na **Figura 1–10**.
4. Nessa última janela, escolha a linguagem, que, obviamente, deve ser C e, então, clique em *Next*. Infelizmente, nem todo iniciante em programação acha isso tão óbvio e escolhe C++, que é outra linguagem (v. **Seção 3.18.1**). Este livro usa C, e não C++, como linguagem de programação. Assim nunca escolha C++ nessa janela.

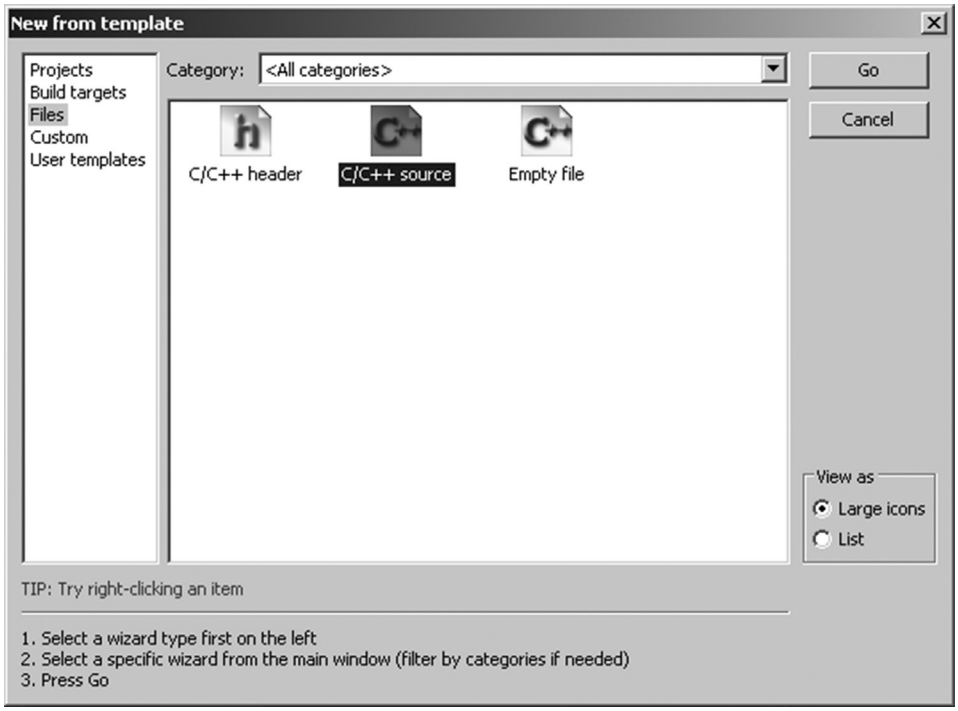

FIGURA 1–9: CRIANDO UM PROGRAMA-FONTE COM CODEBLOCKS 1

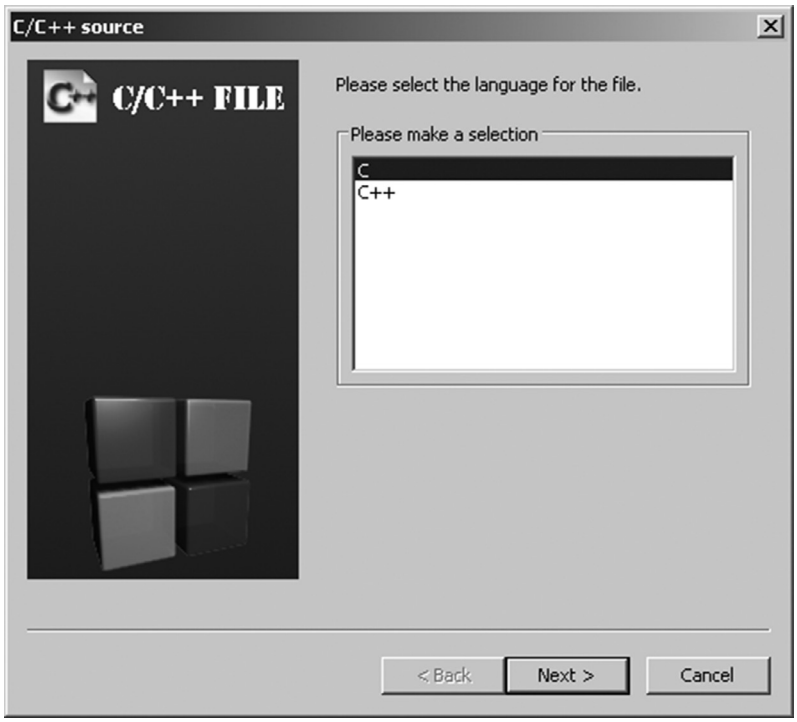

FIGURA 1–10: CRIANDO UM PROGRAMA-FONTE COM CODEBLOCKS 2

5. Na próxima janela apresentada na sequência, você será instado a escolher o nome do programa e o local em que ele será salvo. Proceda normalmente como em qualquer outro programa, mas certifique-se de satisfazer os seguintes requisitos: não inclua espaços em branco no nome do arquivo; utilize `.c` (e não `.cpp`) como extensão do arquivo e salve o arquivo no diretório (pasta) sugerido na Seção 1.6.1.

6. Após selecionar o nome do arquivo e o local onde ele será salvo, clique em *Finish* e você obterá uma janela como aquela mostrada na **Figura 1–11**.

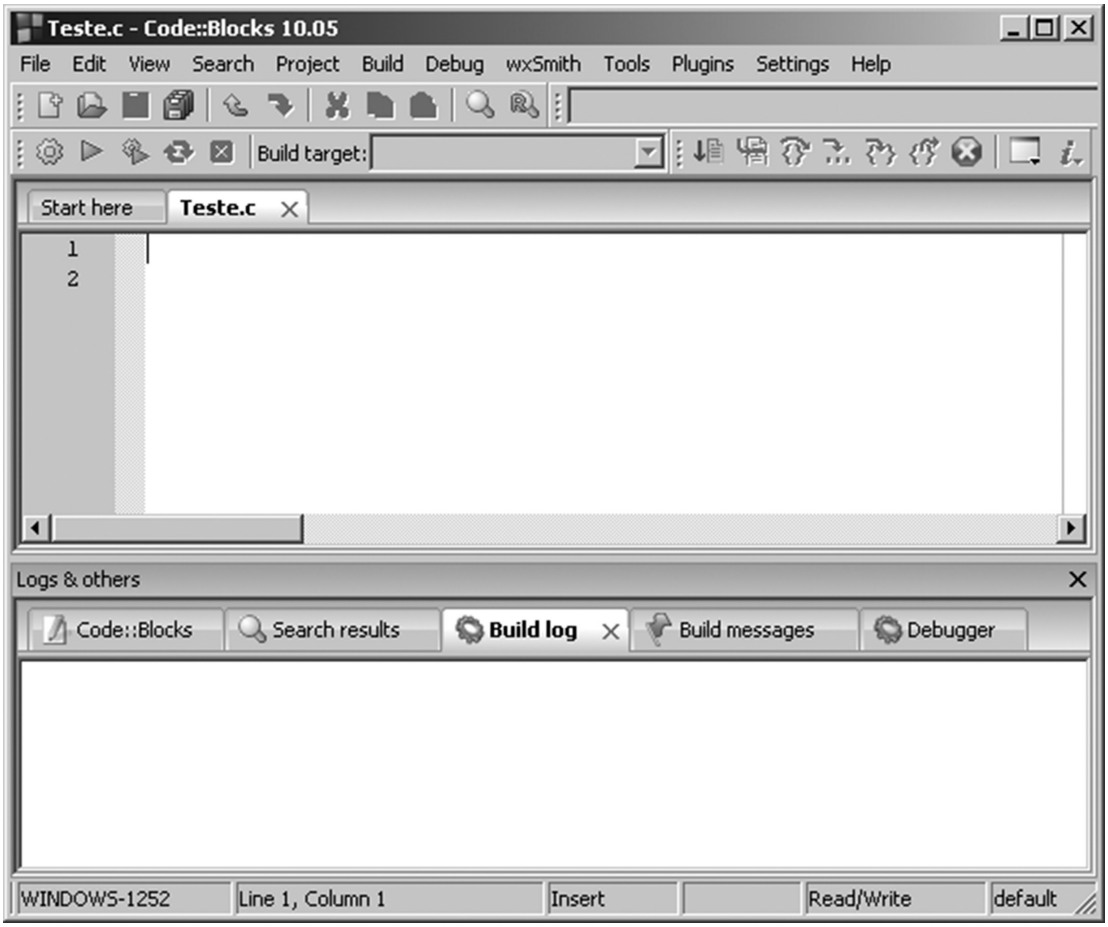

FIGURA 1–11: CRIANDO UM PROGRAMA-FONTE COM CODEBLOCKS 3

O painel maior na **Figura 1–11** é dedicado à edição do programa-fonte que você acaba de criar. O programa responsável por isso é o editor de programas do CodeBlocks.

Conforme descrito na **Seção 1.5**, um editor de programas é um editor de texto com características especiais dirigidas a facilitar a escrita de programas. Para conhecer algumas destas características, digite o seguinte programa no painel de edição e salve-o com o nome `Teste.c`:

```
   /* Meu primeiro programa */
#include <stdio.h>
int main(void)
{
   printf("Este e' meu primeiro programa em C");
   return 0;
}
```

Se você digitar o programa acima exatamente como ele se apresenta, notará que esse editor não é um editor de texto comum como, por exemplo, o bloco de notas do Windows. Quer dizer, em vez de apresentar o texto como ele é, esse editor usa cores diferentes para alguns componentes diferentes. Essa característica é chamada

coloração de sintaxe e tem como objetivo facilitar a identificação visual de componentes de um programa que pertencem a categorias distintas. Contudo, o produto resultante de um editor de programas é o mesmo produzido por um editor de texto comum. Ou seja, se você abrir com um editor de texto comum o arquivo que você acaba de criar, vê-lo-á exatamente como ele é: texto puro; i.e., sem nenhuma formatação.

Para admirar um pouco mais o potencial de um editor de programas execute as seguintes tarefas:

- Logo após a linha contendo um abre-chaves, digite `[ENTER]`. Observe que o cursor de edição não passa para a primeira coluna da próxima linha, como seria o caso se você executasse a mesma ação num editor de texto comum. Ele procede dessa maneira porque ele sabe que o que você irá digitar em seguida estará subordinado à função main() e, por isso, ele cria uma endentação para indicar essa subordinação. Parece que, por enquanto, o editor de programas do CodeBlocks entende mais de programação em C do que você. Mas não se preocupe, pois essa situação logo se reverterá.
- Agora clique numa posição imediatamente antes ou depois do abre-chaves. Você verá que ele e o correspondente fecha-chaves tornam-se simultaneamente sombreados. Se você clicar próximo ao abre-parênteses após *main* ou *printf*, algo equivalente ocorre. Essa facilidade ajuda você a descobrir se existem chaves, parênteses ou colchetes que foram abertos e não foram devidamente fechados ou vice-versa.
- Para concluir essa excitante experiência inicial com um editor de programas, clique no traço que aparece logo após a linha contendo main() na faixa estreita acinzentada à esquerda. Quando você faz isso, parte do texto que era visível desaparece da vista. Mas isso não significa que ele foi removido do arquivo. Esse texto foi apenas encolhido e poderá ser acessado novamente clicando-se no sinal de mais que apareceu em substituição ao sinal de menos que havia antes de o texto ser encolhido. Essa característica é importante quando o programa ora sendo editado é bem grande e você precisa manter o foco temporariamente em apenas algumas partes dele.

Existem outras particularidades do editor de programas do CodeBlocks que podem facilitar sobremaneira a edição de um programa. Entre elas, merecem destaque: complementação de código e uso de abreviações. Outras características interessantes do editor de programas do IDE CodeBlocks serão apresentadas nos **Capítulos 3** e **7**.

1.7.4 Criando um Programa Executável com CodeBlocks

Para transformar um programa-fonte, como aquele digitado na **Seção 1.7.3**, em programa executável, escolha a opção *Build* no menu também denominado *Build*. Alternativamente, você pode pressionar o botão cujo ícone tem forma de engrenagem amarela e que aparece mais à esquerda na barra de ferramentas, como mostra a **Figura 1–12**.

Se você digitou corretamente o programa sugerido na **Seção 1.7.3**, ele será compilado e ligado sem problemas e você será informado sobre isso na aba *Build log* do painel inferior do ambiente CodeBlocks, como mostra a **Figura 1–13**.

Na **Figura 1–13**, as duas primeiras linhas mostram como o compilador (denominado `gcc.exe`) e o linker (também denominado `gcc.exe`) foram executados. Se você configurou o compilador e o linker conforme foi recomendado na **Seção 1.7.2**, as opções de compilação (`-Wall`, `-std=c99` e `-pedantic`) e ligação (`-lleitura` e `-lm`) aparecem nas linhas de comando com as quais essas ferramentas foram executadas.

Se a construção do programa executável foi bem sucedida, você deverá perceber que dois arquivos foram criados no diretório onde se encontra o programa-fonte, como mostra a **Figura 1–14**. Nessa figura, o nome do programa-fonte é `Teste.c`, ele foi compilado no sistema Windows XP e salvo no diretório (pasta) `C:\Programas`.

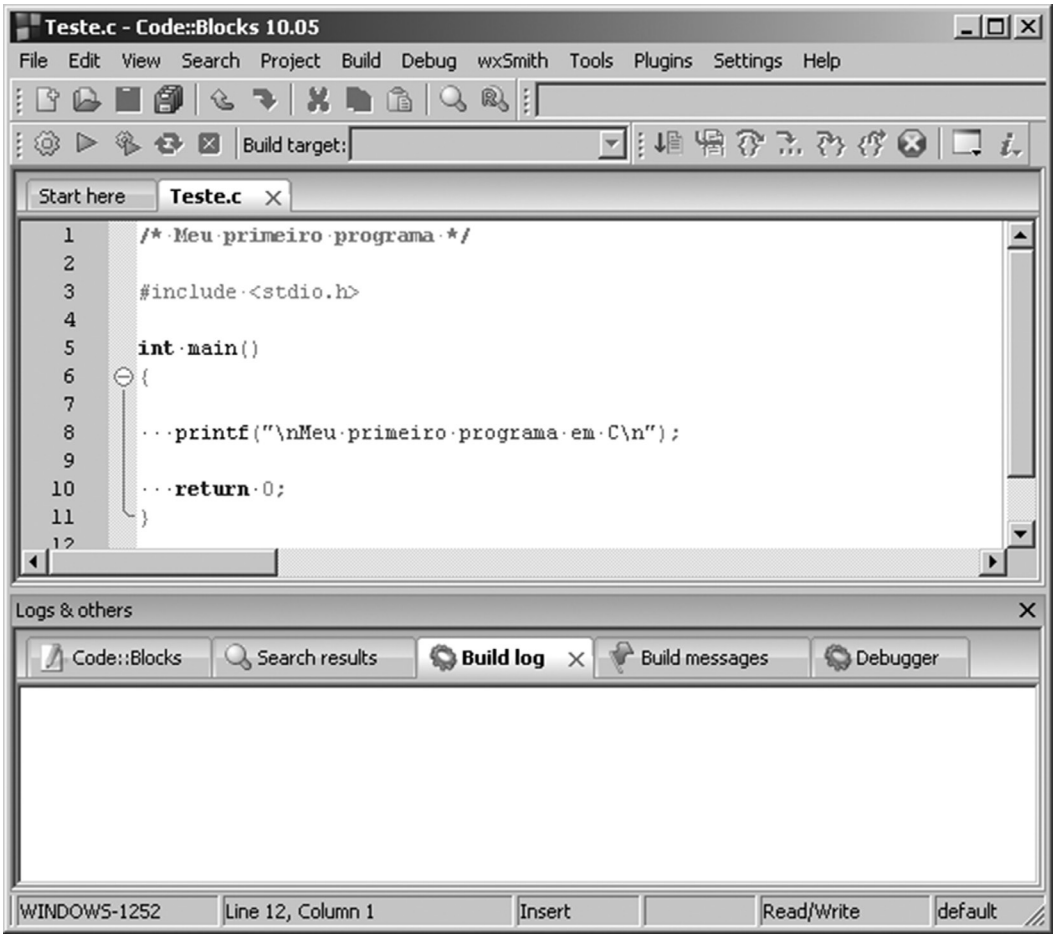

Figura 1–12: Criando um Programa Executável com CodeBlocks 1

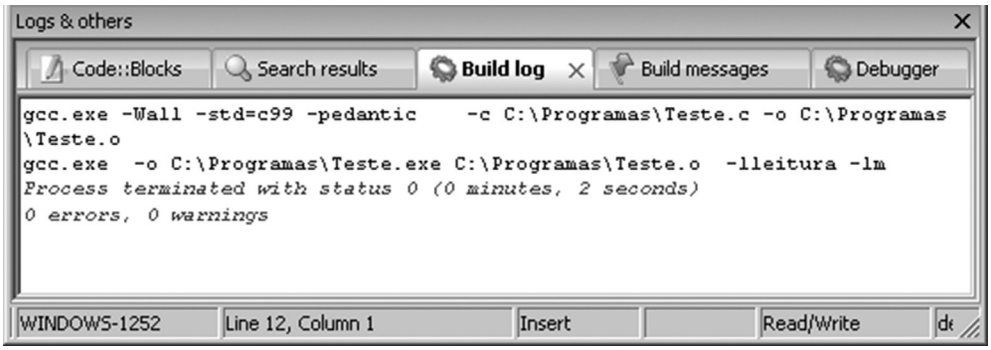

Figura 1–13: Criando um Programa Executável com CodeBlocks 2

Um dos arquivos resultantes do processo de construção do programa executável mostrado na **Figura 1–14** tem o mesmo nome principal do arquivo-fonte e a extensão `.o`. Esse é o arquivo resultante da compilação e, conforme foi discutido na **Seção 1.2.2**, ele contém instruções em linguagem de máquina, mas não é executável. Esse arquivo é usado pelo linker para construir o programa executável, que é o arquivo que tem o mesmo nome principal do arquivo-fonte e a extensão `.exe`. Em sistemas da família Unix (p. ex., Linux), esse arquivo pode ou não ter extensão. Após a criação do programa executável, o arquivo com extensão `.o` não é mais necessário e pode ser removido.

Figura 1–14: Criando um Programa Executável com CodeBlocks 3

Se você cometeu algum erro na digitação do programa-fonte mencionado, seu programa não será compilado e, nesse caso, você obterá uma ou mais mensagens de erro no painel inferior do CodeBlocks, como mostra a **Figura 1–15**.

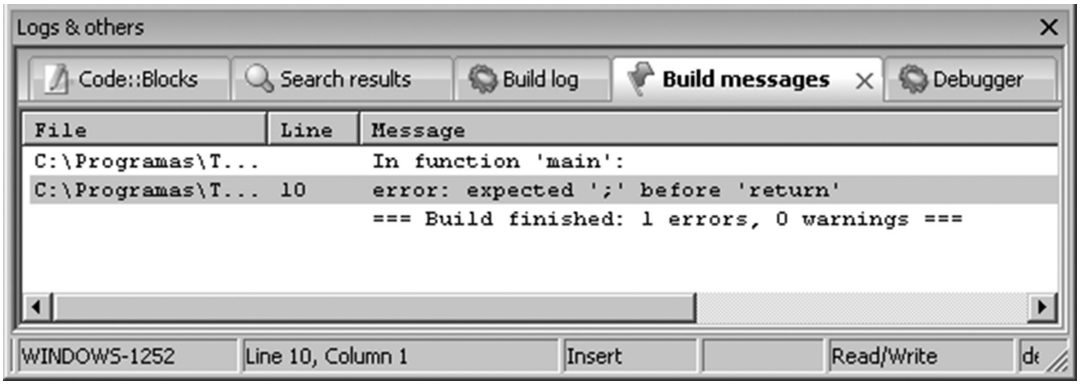

Figura 1–15: Criando um Programa Executável com CodeBlocks 4

O erro ao qual se refere a mensagem apresentada na linha escurecida na **Figura 1–15** foi intencionalmente provocado pela remoção do ponto e vírgula que acompanha a linha contendo return no aludido programa. Esse tipo de erro é denominado *erro de sintaxe* ou *erro de compilação* e será discutido na **Seção 3.18**.

Antes de concluir esta apresentação do IDE CodeBlocks, deve-se observar que, quando um programa consiste em apenas um arquivo-fonte, como é o caso de todos aqueles apresentados neste livro, a opção *Compile current file* do menu *Build*, na realidade, corresponde a compilação e ligação com a consequente obtenção de um arquivo executável. Ou seja, essa opção é equivalente à opção *Build* mencionada.

Do ponto-de-vista prático, não faz sentido apenas compilar (literalmente) um programa constituído por um único arquivo-fonte. Mas, quando um programa é constituído por vários arquivos-fonte, essa operação faz bastante sentido. Por exemplo, suponha que um programa seja constituído por centenas de arquivos-fonte e que compilá-los todos leve um tempo considerável. Utilizando a opção *Compile current file*, você poderá compilar apenas um arquivo-fonte que tenha sido alterado e invocar o linker por meio do comando *Build*; i.e., não será preciso recompilar os demais arquivos antes de efetuar as ligações do programa executável.

O uso da palavra *compilação* em substituição a *compilação e ligação* também ocorre em linguajar cotidiano entre programadores. Quer dizer, quando um programador fala em compilar um programa, na maioria das vezes, na realidade, ele se refere a compilar e efetuar as devidas ligações de modo a obter um programa executável.

1.8 Executando um Programa de Console

Executar um programa baseado em console, como aqueles que se ensinam a desenvolver neste livro, é fácil em sistemas da família Unix/Linux, desde que, evidentemente, você tenha ciência do sistema que está usando, e não apenas de sua interface gráfica. Entretanto, executar um programa baseado em console em sistemas operacionais da família Microsoft Windows não é tão simples quanto parece.

Localize um programa executável construído de acordo com as recomendações encontradas na **Seção 1.7.4** e tente executá-lo como se fosse um programa qualquer do Windows (p. ex., usando um clique duplo sobre o ícone do programa). Se você fizer isso, talvez, você tenha uma desagradável surpresa, pois não verá o resultado esperado, mas apenas observará um breve piscado na tela. Isso não significa que seu programa está incorreto; quer dizer apenas que você não sabe ainda como executá-lo.

Conforme foi visto na **Seção 1.4**, os programas abordados neste livro não se destinam a sistemas da família Microsoft Windows. Em vez disso, eles são dirigidos ao DOS, um antigo sistema operacional que se tornou obsoleto após a introdução do sistema Windows95. Em sistemas operacionais da família Windows, existe emulação do sistema operacional DOS. Isto é, o sistema DOS não mais existe, mas o Windows permite que programas do DOS sejam executados num ambiente que simula o funcionamento desse antigo sistema. Assim, quando você executa um programa do DOS sob Windows, esse sistema age como agiria como se estivesse executando qualquer outro programa: uma janela é aberta (nesse caso uma janela de emulação do DOS), o programa é executado e, finalmente, a janela é fechada quando o programa encerra. Portanto, para que você visualize algum resultado da execução de um programa dirigido ao DOS nessas circunstâncias, é necessário que o programa seja executado por um longo período ou que sua execução seja interrompida (sem ser encerrada).

Essa situação parece ter propagado a ideia de que qualquer programa escrito em C deve conter uma instrução tal como:

```
getchar(); /* Essa opção é ruim */
```

ou

```
system("PAUSE"); /* Essa opção é pior: PAUSE não é portável */
```

Não há mal nenhum em, circunstancialmente, utilizar, por exemplo, getchar() com o objetivo de evitar que a janela de console desapareça antes que se tenha oportunidade de examinar o que um programa escreve. O mal é achar que incluir tal instrução é essencial num programa em C em qualquer situação.

Programadores que usam consistentemente tais instruções com o objetivo de *parar a tela* acham que, sem uma delas, seria impossível ler no terminal aquilo que o programa escreve. Todavia, a incapacidade de visualização de resultados escritos na tela pelo programa nessas circunstâncias é em virtude do simples fato de o sistema Windows fechar a janela de console DOS quando a execução do programa é encerrada, se essa janela foi aberta apenas para a execução do programa. Uma forma de contornar esse problema consiste em executar o prompt do DOS, navegar até o diretório onde se encontra seu programa e, na linha de comando, digitar o nome do programa executável. Assim, você verá o resultado do programa, como mostrado na **Figura 1–16**, sem que a janela na qual ele é executado seja fechada automaticamente.

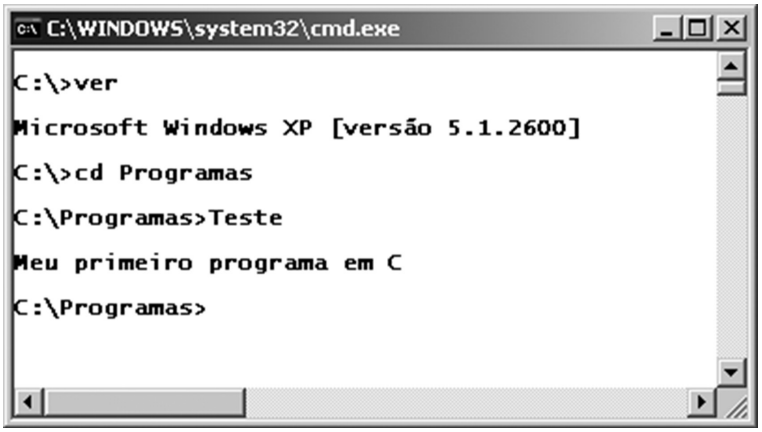

FIGURA 1–16: EXECUÇÃO DE UM PROGRAMA NO SISTEMA DOS/WINDOWS

Uma dica que lhe livra do incômodo de ter que digitar o nome do programa no prompt do DOS é seguir os seguintes passos:

1. Abra a pasta que contém o arquivo executável que você deseja executar e clique sobre o ícone desse arquivo (cuja extensão deve ser `.exe`).
2. Pressione `[F2]`, para selecionar o nome desse arquivo.
3. Pressione `[CTRL]+[C]` para copiar o nome do arquivo.
4. Passe para a janela de prompt do DOS, que deverá estar aberta no diretório que contém o referido arquivo executável.
5. Clique sobre a janela do DOS com o botão direito do mouse e, no menu que aparece, escolha a opção colar. Executando essa operação, você terá copiado o nome do programa executável no prompt do DOS.
6. Pressione `[ENTER]` e seu programa será executado.

Após executar seu programa e estar satisfeito com os resultados que ele apresenta, você poderá remover o arquivo executável, pois, afinal, você já sabe como construí-lo novamente sem muito trabalho. Em resumo, os únicos arquivos que você *não deve* apagar são os arquivos-fonte. Isto é, ao contrário, você deve efetuar frequentemente cópias (backups) desses arquivos e mantê-las em segurança.

1.9 Como Aprender a Programar

Nenhuma linguagem de programação pode ser aprendida apenas estudando-se material escrito ou assistindo-se aulas. Isto é, um bom livro de programação ou um bom expositor pode ajudar o aprendiz a entender as construções de uma determinada linguagem e a convencê-lo a adotar boas práticas de programação. Mas, nem um nem outro podem ser diretamente responsabilizados pela formação de um bom ou mau programador. Programação é algo que se aprende apenas com muita prática, perseverança e durante um período considerável de tempo. Portanto seja paciente porque esse processo requer muito tempo de aprendizagem e, por favor, nunca acredite em livros que prometem ensiná-lo a programar em 24 horas!

A seguir, serão apresentadas sugestões para uma melhor assimilação das técnicas de programação apresentadas neste livro[4].

- **Estude os programas minimalistas**. Esses programas são curtos e procuram enfocar apenas uma construção comum em programação ou uma característica da linguagem C. Em virtude da sua relativa simplicidade, provavelmente, você não deverá ter dificuldade em entender cada um desses programas.

[4] Alguns desses ensinamentos são inspirados no magnífico trabalho de George Pólya: *How to Solve It* (v. **Bibliografia**).

Mas, se for o caso, releia a parte do texto que antecede o programa e procure entender aquilo que o programa pretende demonstrar. Então, observe atentamente o resultado do programa e tente compreender como ele é produzido. Se ainda não conseguir entendê-lo, explore-o ativamente conforme descrito a seguir. Não esqueça que os arquivos-fonte desses programas estão disponíveis para download no site ***www.ulysseso.com/ip***.

- **Explore ativamente os exemplos de programação**. Os programas que fazem partes das seções intituladas **Exemplos de Programação** são relativamente complexos, visto que podem enfocar várias técnicas de programação ao mesmo tempo. Para entender cada um deles, sua primeira atitude deve ser obter uma clara compreensão do problema que o programa sob escrutínio resolve. Se você não conseguir entender o enunciado de um dado problema, não adianta insistir e tentar compreender o programa que o soluciona. Para entender bem o que um programa faz, compile-o e, então, execute-o como um usuário comum. Então, leia o programa apresentado no livro e estude sua análise. Se não assimilar algum segmento do programa, abra o respectivo arquivo-fonte, altere esse segmento, compile o programa e veja como ele se comporta (i.e., verifique qual é o novo resultado que ele produz). Se um programa deixar de funcionar após tê-lo modificado, tente entender o porquê. Testando suas próprias ideias a respeito de um dado programa, você poderá obterá um melhor entendimento sobre como ele funciona. Lembre-se que os arquivos-fonte desses programas estão disponíveis para download no site ***www.ulysseso.com/ip***.
- **Use auxílio visual para acompanhar algoritmos e programas**. Alguns algoritmos são difíceis de ser acompanhados mentalmente, de forma que representações gráficas de variáveis envolvidas nesses algoritmos podem ajudar bastante o entendimento deles. Esse artifício funciona do seguinte modo: usando-se lápis, papel e borracha desenham-se retângulos representando as variáveis de interesse; o interior de cada retângulo representa o conteúdo de uma variável, que tem seu nome escrito em algum local próximo ao retângulo; então, quando uma variável tem seu valor alterado, o retângulo associado a ela tem seu interior atualizado de modo a refletir essa mudança de valor. Valores assumidos por parâmetros também podem ser acompanhados da mesma forma (v. exemplo na **Seção 5.5**). Outrossim, esse recurso visual é particularmente importante no caso de algoritmos seguidos por programas que lidam com arrays (v. exemplos no **Capítulo 8**) e strings (v. exemplo na **Seção 9.5.9**).
- **Resolva os exercícios de revisão**. Muitos exercícios de revisão solicitam que você encontre falhas em pequenos programas. Dedique algum tempo a esses programas até descobrir por que eles não funcionam. Alguns deles incorporam erros sutis e difíceis de serem descobertos. Em tal caso, digite o programa e tente compilá-lo. Se o erro for de sintaxe, evidentemente, o compilador o indicará. Se o programa for compilado, talvez o compilador apresente uma ou mais mensagens de advertência, que o ajudarão a descobrir o erro. Se o compilador não apresentar nenhuma mensagem de erro ou advertência, execute o programa e o erro certamente manifestar-se-á em algum instante. Exercícios que requerem que você identifique erros em programas provêm um importante treinamento, pois, mais adiante, quando você se deparar com erros similares em programas mais complexos, estará habilitado a corrigi-los com mais facilidade.
- **Tente resolver todos os exercícios de programação**. Para resolver os exercícios de programação propostos ao final de cada unidade, siga rigorosamente os roteiros propostos para construção de algoritmos e programas apresentados nos **Capítulos 2** e **3**, principalmente quando estiver lidando com problemas mais complexos. Um bom programa reflete um bom projeto de solução para o respectivo problema. Portanto pense bastante sobre a linha de solução (algoritmo) a ser seguida antes de começar a codificar seu programa. Se você não for capaz de escrever um algoritmo que resolve um determinado

problema, é provável que você não saiba como resolvê-lo, ou saiba como resolvê-lo mas é incapaz de descrever sua solução.

- **Abstenha-se de problemas que requerem conhecimento que você não possui**. Programação abrange virtualmente todas as áreas de conhecimento. Ensinar a programar consiste em educar alguém a exprimir seu conhecimento sobre a resolução de um problema por meio de instruções que serão convertidas num programa. Logo um curso de introdução à programação não deve ser encarado como um curso sobre resolução de problemas aleatoriamente extraídos de qualquer domínio. Assim, por exemplo, se um aprendiz não sabe criar um algoritmo que verifica se três números reais constituem os lados de um triângulo, isso não significa que ele não sabe construir um algoritmo que expresse seu conhecimento. Isso pode apenas exprimir que ele não tem conhecimento específico sobre o problema para resolvê-lo (nesse exemplo, conhecimento sobre desigualdade triangular). Portanto, se você não possui conhecimento prévio para acompanhar um dado exemplo ou resolver um exercício proposto de programação não deve insistir nem desanimar. Neste livro, existem inúmeros exemplos e exercícios envolvendo diversas áreas de conhecimento, de forma que se alguns forem saltados não haverá prejuízo considerável para o aprendiz.
- **Entenda o enunciado ou desista**. Antes de abordar um problema que você julga que entendeu claramente o enunciado, certifique-se que sua suposição realmente é legítima. Quando estiver convencido disso e, mesmo assim, não conseguir encontrar uma solução para o problema original, tente resolver um modelo simples desse problema. Por exemplo, suponha que o problema em questão consiste em resolver equações de segundo grau; então, antes de tentar resolver o problema por completo, tente resolver um caso particular de equação do segundo grau (p. ex., usando coeficientes constantes e sem entrada de dados). Assim procedendo, você poderá adquirir insight para resolver o problema completo.
- **Não seja ansioso ou apressado**. Resista à tentação de querer resolver todos os problemas diretamente no computador. Aos poucos, você estará apto a proceder assim com a maioria dos problemas do nível daqueles propostos neste livro introdutório, mas mesmo programadores experientes precisam afastar-se do computador e debruçar-se manualmente sobre alguns problemas para planejar suas soluções. Muitos aprendizes de programação não resistem à ansiedade e costumam começar a codificar tão logo se deparam com um problema para resolver. Quer dizer, para eles, resolução de problemas via computador significa criar programas e corrigir erros indicados pelo compilador ou que se manifestam durante a execução do programa. Se você se habituar a essa abordagem de construir e consertar (ou tentativa e erro), terá sérias dificuldades quando tiver que lidar com problemas mais complexos.
- **Adote um estilo de programação coerente e bem fundamentado**. Normas de estilo não se referem apenas a questões estéticas; elas podem lhe livrar de erros de programação que poderiam ser evitados, como mostra o **Capítulo 4**. Logo tente resolver todos os exercícios de programação usando um bom estilo de programação, que é alcançado por meio de experiência e prática de programação. Enquanto não adquire experiência e capacidade de discernir entre o que é certo e o que não é recomendado, evite examinar material sobre programação encontrado na internet. Democrática como é, a internet permite a qualquer um submeter qualquer coisa e, infelizmente, a maior parte do material sobre programação publicado na internet é de má qualidade. Mas, se você aderir aos conselhos básicos de estilo de programação apresentados neste livro, seu estilo estará bem fundamentado. Se você estudar sob supervisão, pode adotar outro estilo desde que o submeta à apreciação de seu supervisor. Mas, se você estiver estudando sem supervisão, é aconselhável adotar o estilo apresentado aqui, pois ele reflete muitos anos de experiência de programação em C. Em qualquer caso, existe apenas uma regra em estilo que deve ser rigorosamente observada:

Recomendação *Mantenha seus programas claros, concisos e simples.*

- **Comente, comente, comente...** Comentar programas ajuda a organizar o raciocínio que norteia a construção de cada programa e a manter esse salutar hábito. Mesmo que um programa seja usado apenas por você, se ele não tiver boa legibilidade ou não for bem documentado, talvez você tenha dificuldade de entendê-lo mais adiante. Pode parece estranho, mas isso ocorre com mais frequência do que você possa imaginar. Logo, você verá que muitos programas sem comentários que pareciam óbvios quando você os escreveu deixarão de ser evidentes algum tempo depois. Portanto sempre comente seus programas, mesmo que você seja a única pessoa que os lerá.
- **Pratique, pratique, pratique...** A principal diferença entre um programador experiente e um iniciante é que o primeiro reúne um grande repertório de problemas resolvidos que podem ser recobrados de sua memória quando ele se depara com um novo problema similar a algum problema conhecido. Por isso, muito raramente, um programador experiente começa a escrever algum programa a partir de nada, como fazem os iniciantes. Para um programador experiente, sempre há um programa que ele há já escreveu que pode servir de base para um novo programa, seja por meio de alterações, acréscimos ou subtrações de código. Concluindo, quanto mais você praticar, mais estará apto a beneficiar-se com o conhecimento adquirido na construção de novos programas. Logo pratique, pratique, pratique...
- **Não reinvente a roda**. Como foi visto no último parágrafo, em programação, muitos problemas são recorrentes, de modo que, com alguma experiência, um programador é capaz de identificar semelhanças entre um novo problema de programação e um problema já resolvido e incorporar partes da solução do problema conhecido na solução do novo problema. Mesmo quando dois programas parecem ser completamente diferentes, sempre há algo de um programa que se pode usar em outro por meio de técnicas simples, como se demonstra no **Capítulo 7**.

1.10 Exemplo de Programação

1.10.1 Testando a Instalação de LeituraFacil

Problema: Teste a instalação da biblioteca LeituraFacil prescrita na **Seção 1.6.4** digitando o programa a seguir conforme foi descrito na **Seção 1.7.3**. Então, crie um programa executável conforme foi ensinado na **Seção 1.7.4**, execute-o conforme sugerido na **Seção 1.8** e, finalmente, utilize-o como um usuário comum o faria.

Solução:

```
#include <stdio.h>
#include "leitura.h"

int main(void)
{
   int    i, c;
   double f;

   printf("\nDigite um caractere > ");
   c = LeCaractere();

   printf("\n>> Caractere digitado: %c\n", c);

   printf("\nDigite um numero inteiro > ");
   i = LeInteiro();
```

```
   printf("\n>> Numero inteiro digitado: %d\n", i);
   printf("\nDigite um numero real > ");

   f = LeReal();
   printf("\n>> Numero real digitado: %f\n", f);

   return 0;
}
```

Exemplo de execução do programa:

```
Digite um caractere > A
>> Caractere digitado: A

Digite um numero inteiro > 21
>> Numero inteiro digitado: 21

Digite um numero real > 2.5
>> Numero real digitado: 2.500000
```

Análise: O exemplo acima foi construído para que você teste sua instalação da biblioteca LeituraFacil, bem como para situá-lo no contexto de programação oferecendo uma noção daquilo que você deverá aprender. Portanto considere esse exemplo como cenas dos próximos capítulos.

1.11 Exercícios de Revisão

Um Breve Histórico de Linguagens de Programação (Seção 1.1)

1. O que é linguagem de máquina?
2. Por que programas escritos em linguagem de máquina não são portáveis?
3. Quais foram as facilidades introduzidas em programação com o surgimento da linguagem assembly?
4. O que é uma linguagem de baixo nível?
5. (a) O que é uma linguagem de alto nível? (b) Cite três exemplos de linguagens de alto nível.
6. Em que aspectos linguagens de baixo nível diferem de linguagens de alto nível?
7. Em que aspectos linguagens de máquina assemelham-se à linguagens assembly?
8. (a) O que é uma variável em programação? (b) O que é uma variável simbólica?
9. Cite vantagens obtidas com o uso de linguagens de alto nível em relação a linguagens de baixo nível.
10. Por que a linguagem assembly ainda é usada em programação nos dias atuais?

Tradutores (Seção 1.2)

11. Por que assembler é o tipo de tradutor mais simples que existe?
12. Um tradutor assembler para um dado processador pode ser melhor do que outro assembler para o mesmo processador? Explique.
13. (a) Um computador pode entender um programa escrito em C? (b) O que é necessário para traduzir um programa escrito em C num programa escrito em linguagem de máquina (código binário)?
14. (a) O que é compilação? (b) O que é interpretação? (c) Em que diferem esses conceitos?
15. O que é um compilador?
16. (a) O que é um programa-fonte? (b) O que é um programa-objeto? (c) O que é um programa executável?
17. Quem é responsável pela criação de cada tipo de arquivo a seguir?
    (a) Programa-fonte

(b) Programa-objeto
(c) Programa executável

18. Suponha que você tem dois compiladores que traduzem programas escritos na linguagem de alto nível X para a linguagem de máquina do processador Y. É possível que um deles seja melhor do que o outro? Explique.

19. Um programa compilado é necessariamente executável? Explique.

20. Qual é o papel de um linker no processo de desenvolvimento de um programa?

21. O que é um interpretador e como ele funciona?

22. Por que um programa interpretado leva mais tempo para ser executado do que um programa compilado equivalente?

23. Compare compiladores e interpretadores em termos de vantagens e desvantagens de cada tipo de tradutor.

Qualidades de um Bom Programa (Seção 1.3)

24. Defina as seguintes propriedades desejáveis de um programa de boa qualidade:
(a) Legibilidade
(b) Manutenibilidade
(c) Portabilidade
(d) Eficiência
(e) Reusabilidade
(f) Robustez
(g) Usabilidade
(h) Confiabilidade

25. Um programa escrito numa linguagem de baixo nível tem sempre pouca legibilidade? Explique.

26. Um programa escrito numa linguagem de alto nível tem necessariamente boa legibilidade? Explique.

27. O que influencia a portabilidade de um programa?

28. O que significa um bom estilo de programação?

29. Por que muitas vezes é tão difícil, ou mesmo impossível, criar um programa que apresente a máxima eficiência possível utilizando uma linguagem de alto nível?

30. Na **Seção 1.2.3**, afirma-se que o uso de interpretadores facilita a portabilidade de programas-fonte. Por que o mesmo raciocínio não se aplica a compiladores?

31. Quando dois programas são considerados funcionalmente equivalentes?

32. Por que programadores em estágio inicial de aprendizagem não devem se preocupar em obter programas com a máxima eficiência possível?

Programas Interativos Baseados em Console (Seção 1.4)

33. (a) O que é um programa interativo? (b) Cite três exemplos de programas interativos. (c) Cite dois exemplos de programas que não são interativos.

34. (a) O que é um console (ou terminal)? (b) Que dispositivos periféricos constituem um console?

35. O que é um programa baseado em console?

Ambientes Integrados de Desenvolvimento (Seção 1.5)

36. O que é um ambiente integrado de desenvolvimento (IDE)?

37. Quais são as vantagens obtidas com o uso de um editor de programas em detrimento ao uso de um editor de texto comum na construção de programas.

38. (a) O que é coloração de sintaxe? (b) Qual é sua utilidade na construção de um programa?

39. Por que o programador deve evitar alterações frequentes na configuração de coloração de sintaxe de um editor de programas?

40. Qual é o papel desempenhado por um programa carregador (*loader*)?

41. O que é um depurador?

Instalação de um Ambiente de Trabalho (Seção 1.6)

42. O que é uma cadeia de ferramentas de desenvolvimento (*toolchain*)?

43. Qual é a diferença entre IDE e *toolchain*? [**Sugestão**: A diferença é um pouco sutil e alguns consideram os dois conceitos iguais. Mas, de acordo com as definições apresentadas neste capítulo há diferenças.]

44. (a) O que é um carregador de programas? (b) O IDE CodeBlocks utiliza carregador de programas?

45. Como é denominado o depurador que acompanha o pacote MinGW?

46. O que é uma biblioteca de componentes e qual é sua importância para o programador?

47. Quais são as vantagens oferecidas pelo pacote MinGW?

48. Que opção do compilador GCC é utilizada para a geração do maior número possível de mensagens de erro?

49. Qual é o papel de um linker (editor de ligações) no processo de desenvolvimento de um programa?

50. (a) O que é ligação no processo de construção de um programa executável? (b) Qual é a diferença entre compilação e ligação?

51. (a) O que é a biblioteca LeituraFacil? (b) Que vantagens ela oferece ao aprendiz de programação?

Utilizando CodeBlocks (Seção 1.7)

52. (a) O que é CodeBlocks? (b) Que vantagens ele oferece ao programador?

53. Como se cria um programa-fonte no IDE CodeBlocks?

54. Tendo disponível um programa-fonte, como se cria um programa executável correspondente no IDE CodeBlocks?

55. Se as opções *Compile current file* e *Build* do menu *Build* do IDE CodeBlocks funcionam do mesmo modo, por que, afinal, existem essas duas opções (e não apenas uma delas)?

Executando um Programa de Console (Seção 1.8)

56. Qual é a utilidade da extensão de interface denominada *Abrir janela de comando aqui* (ou *Open command window here*) dos sistemas da família Windows? [Esta questão diz respeito a Windows XP e a versões de Windows anteriores a esse sistema. Se você não usa tal sistema, não precisa preocupar-se com esta questão.]

57. Por que maus programadores de C sempre incluem `getchar()` ou `system("PAUSE")` em seus programas?

Como Aprender a Programar (Seção 1.9)

58. Descreva as sugestões para melhorar o aprendizado de programação apresentadas neste capítulo.

59. O que são programas minimalistas?

60. O que significa *explorar ativamente exercícios de programação*?

61. Como se pode obter auxílio visual para facilitar o acompanhamento de algoritmos e programas?

62. Por que o iniciante em programação deve evitar a tentação de resolver problemas diretamente no computador?

63. Qual é a importância da prática de programação para o iniciante nessa atividade?

1.12 Exercícios de Programação

EP1.1 Execute o CodeBlocks e selecione a opção *Editor* no menu *Settings*. Então, assegure que o editor é configurado com as opções mostradas na figura a seguir.

EP1.2 Execute o CodeBlocks e selecione a opção *Editor* no menu *Settings*. Em seguida, clique no ícone intitulado *Margins and caret*. Então, assegure que o editor é configurado com as opções mostradas na figura a seguir.