Introdução à Programação em C

C Como Programar, Sexta Edição

Objetivos

- A escrever programas de computador em C.
- A usar instruções simples de entrada e de saída.
- A usar os tipos de dados fundamentais.
- Conceitos de memória do computador.
- A usar operadores aritméticos.
- A ordem de precedência dos operadores aritméticos.
- A escrever instruções simples de tomada de decisão.

2.1 Introdução

- A linguagem C facilita uma abordagem estruturada e disciplinada do projeto de um programa de computador.
- Neste capítulo, introduzimos a programação em C e apresentamos vários exemplos que ilustram muitas características importantes de C.
- Nos capítulos 3 e 4, apresentaremos uma introdução à programação estruturada em C.

2.2 Um programa em C simples: imprimindo uma linha de texto

- Podemos começar considerando um programa em C simples, que imprime uma linha de texto.
- O programa e sua saída na tela são mostrados na Figura 2.1.

```c
/* Figura 2.1: fig02_01.c
   Primeiro programa em C */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   printf( "Bem-vindo a C!\n" );

   return 0; /* indica que o programa terminou com sucesso */
} /* fim da função main */
```

```
Bem-vindo a C!
```

Figura 2.1 ■ Primeiro programa em C.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- As linhas 1 e 2
  - `/* Figura 2.1: fig02_01.c`
    `Primeiro programa em C */`
- começam cada uma com /* e terminam com */, o que indica que essas duas linhas são um comentário.
- Você insere comentários para documentar programas e melhorar a legibilidade deles.
- Os comentários não levam o computador a executar nenhuma ação quando o programa está sendo executado.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Os comentários são ignorados pelo compilador C e não geram nenhum tipo de código-objeto em linguagem de máquina.
- O comentário simplesmente descreve o número da figura, o nome do arquivo e o propósito do programa.

Erro comum de programação 2.1
*Esquecer de encerrar um comentário com */.*

Erro comum de programação 2.2
*Iniciar um comentário com os caracteres */ ou encerrá-lo com /*.*

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- C99 também inclui os comentários de linha única // em que tudo, desde // até o fim da linha, é um comentário.
- Eles podem ser usados como comentários isolados, em suas próprias linhas, ou como comentários de fim de linha, à direita de uma linha de código parcial.
- Alguns programadores preferem os comentários com //, pois são mais curtos e eliminam os erros comuns de programação que ocorrem com o uso de /* */.
- A linha 3
  - `#include <stdio.h>`
- é uma diretiva do pré-processador C.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- As linhas que começam com # são verificadas pelo pré-processador antes de o programa ser compilado.
- A linha 3 diz ao pré-processador que inclua no programa o conteúdo do cabeçalho-padrão de entrada/saída (<stdio.h>).
- Esse cabeçalho contém informações usadas pelo compilador quando a compilação solicita as funções da biblioteca-padrão de entrada/saída, como `printf`.
- A linha6
  - `int main( void )`
- faz parte de todo programa em C.
- Os parênteses depois de main indicam que main é um bloco de construção de programa chamado de função.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Os programas em C contêm uma ou mais funções, uma das quais deve ser a `main`.
- Todos os programas em C começam a executar na função `main`.
- A palavra-chave `int` à esquerda de `main` indica que `main` 'retorna' um valor numérico inteiro.
- Explicaremos o que 'retornar um valor' quer dizer no caso de uma função quando demonstrarmos como você pode criar suas próprias funções no Capítulo 5.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Por enquanto, simplesmente inclua a palavra-chave `int` à esquerda de `main` em cada um de seus programas.
- As funções também podem receber informações quando são chamadas para execução.
- O `void` entre parênteses significa que `main` não recebe informação nenhuma.

Boa prática de programação 2.1

Toda função deveria começar com um comentário e a descrição da finalidade da função.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- A chave à esquerda, `{`, inicia o corpo de qualquer função (linha 7).
- Uma chave à direita correspondente, `}`, encerra o corpo de cada função (linha 11).
- Esse par de chaves e a parte do programa entre as chaves é o que chamamos de bloco. O bloco é uma unidade
- de programa importante em C.
- Linha 8
  - `printf( "Bem-vindo a C!\n" );`
- instrui o computador a realizar uma ação, a saber, imprimir na tela a string de caracteres marcada pelas aspas.
- Uma string também é chamada de mensagem, ou de literal.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- A linha inteira, incluindo printf, seu argumento dentro dos parênteses e o ponto e vírgula (;), é denominada instrução (ou comando).
- Cada instrução precisa terminar com um ponto e vírgula (também conhecido como terminador de instrução).
- Quando a instrução `printf` anterior é executada, ela imprime a mensagem `Bem-vindo a C!` na tela.
- Na instrução `printf`, os caracteres normalmente são impressos exatamente como eles aparecem entre as aspas.
- Note, porém, que os caracteres `\n` não são exibidos na tela.
- A barra invertida (\) é chamada de caractere de escape.
- Ela indica que `printf` deve fazer algo
- fora do comum.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Quando uma barra invertida é encontrada em uma string de caracteres, o compilador examina o próximo caractere e o combina com a barra invertida para formar uma sequência de escape.
- A sequência de escape `\n` significa nova linha (*newline*).
- Quando uma sequência de nova linha aparece na string de saída por um `printf`, a nova linha faz com que o cursor se posicione no início da próxima linha na tela.
- Algumas sequências de escape comuns são listadas na Figura 2.2.

Sequência de escape	Descrição
\n	Nova linha. Posiciona o cursor da tela no início da próxima linha.
\t	Tabulação horizontal. Move o cursor da tela para a próxima posição de tabulação.
\a	Alerta. Faz soar o alarme do sistema.
\\	Barra invertida. Insere um caractere de barra invertida em uma string.
\"	Aspas. Insere um caractere de aspas em uma string.

Figura 2.2 ■ Algumas sequências comuns de escape.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Visto que a barra invertida tem um significado especial em uma string, ou seja, o compilador a reconhece como um caractere de escape, usamos uma dupla barra invertida (\\) para colocar uma única barra invertida em uma string.
- A impressão do caractere de aspas também apresenta um problema, pois as aspas marcam o limite de uma string — mas essas aspas não são impressas.
- Usando a sequência de escape`\"` em uma string a ser
- enviada para a saída por `printf`, indicamos que `printf` deverá exibir o caractere de aspas.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Linha 10
  - `return 0; /* indica que o programa foi concluído com sucesso */`
- é incluída no fim de toda função `main`.
- A palavra-chave return de C é um dos vários meios que usaremos para sair de uma função.
- Quando a instrução return for usada no fim de main, como mostramos aqui, o valor 0 indica que o programa foi concluído com sucesso.
- A chave à direita, } (linha 12), indica o final de `main`.

Boa prática de programação 2.2
Inclua um comentário na linha que contém a chave à direita, }, e que fecha cada função, inclusive a main.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Dissemos que `printf` faz com que o computador realize uma ação.
- Quando qualquer programa é executado, ele realiza uma série de ações e toma decisões.
- Ao final deste capítulo, discutiremos a tomada de decisão. No Capítulo 3, discutiremos esse modelo de ação/decisão da programação em detalhes.

Erro comum de programação 2.3
Em um programa, digitar o nome da função de saída, printf*, como* print.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- As funções da biblioteca-padrão, como `printf` e `scanf` não fazem parte da linguagem da programação em C.
- Por exemplo, o compilador não pode encontrar um erro de ortografia em `printf` ou `scanf`.
- Quando compila uma instrução `printf`, ele simplesmente oferece espaço no programa-objeto para uma 'chamada' à função da biblioteca.
- Mas o compilador não sabe onde estão as funções da biblioteca; o linker sabe.
- Quando o linker é executado, ele localiza as funções da biblioteca e insere as chamadas apropriadas nessas funções do programa-objeto.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Agora, o programa-objeto está completo e pronto para ser executado.
- Por esse motivo, o programa vinculado é chamado executável.
- Se o nome da função estiver escrito de forma errada, é o linker que acusará o erro, pois não poderá combinar o nome no programa em C com o nome de qualquer função conhecida nas bibliotecas.

Boa prática de programação 2.3

Recue em um nível o corpo inteiro de cada função (recomendamos três espaços) dentro das chaves que definem o corpo da função. Esse recuo destaca a estrutura funcional dos programas e ajuda a torná-los mais fáceis de serem lidos.

Boa prática de programação 2.4

Estabeleça uma convenção de sua preferência para o tamanho dos recuos; depois, aplique uniformemente essa convenção. A tecla de tabulação pode ser usada para criar recuos, mas pontos de tabulação podem variar. Recomendamos que sejam usados três espaços por nível de recuo.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- A função `printf` pode imprimir `Bem-vindo a C!` de vários modos.
- Por exemplo, o programa da Figura 2.3 produz a mesma saída do programa da Figura 2.1.
- Isso funciona porque cada `printf` reretoma a impressão do ponto em que a instrução `printf` anterior havia parado.
- O primeiro `printf` (linha 8) imprime `Bem-vindo` seguido por um espaço, e o segundo `printf` (linha 9) começa a imprimir na mesma linha, imediatamente após o espaço.

```c
/* Figura 2.3: fig02_03.c
   Imprimindo em uma linha com duas instruções printf */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   printf( "Bem-vindo " );
   printf( "a C!\n" );

   return 0; /* indica que o programa foi concluído com sucesso */
} /* fim da função main */
```

```
Bem-vindo a C!
```

Figura 2.3 ■ Imprimindo em uma linha com duas instruções `printf`.

2.2 Um programa em C simples: imprimindo uma linha de texto (Cont.)

- Um único `printf` pode imprimir múltiplas linhas usando caracteres de uma nova linha, como na Figura 2.4.
- Sempre que a sequência de escape `\n` (nova linha) é encontrada, a saída continua no início da próxima linha.

```
1  /* Figura 2.4: fig02_04.c
2     Imprimindo várias linhas com uma única instrução printf */
3  #include <stdio.h>
4
5  /* função main inicia execução do programa */
6  int main( void )
7  {
8   printf( "Bem-vindo\na\nC!\n" );
9
10    return 0; /* indica que o programa foi concluído com sucesso */
11 } /* fim da função main */
```

```
Bem-vindo
a
C!
```

Figura 2.4 ■ Imprimindo várias linhas com um único `printf`.

2.3 Outro programa em C simples: somando dois inteiros

- Nosso próximo programa utiliza a função `scanf` da biblioteca-padrão para obter dois inteiros digitados no teclado pelo usuário, calcula a soma desses valores e imprime o resultado usando `printf`.
- [No diálogo de entrada/saída da Figura 2.5, enfatizamos os números inseridos pelo usuário em **negrito**.]

```c
/* Figura 2.5: fig02_05.c
   Programa de adição */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   int inteiro1; /* primeiro número a ser informado pelo usuário */
   int inteiro2; /* segundo número a ser informado pelo usuário */
   int soma; /* variável em que a soma será mantida */

   printf( "Digite o primeiro inteiro\n" ); /* prompt */
   scanf( "%d", &inteiro1 ); /* lê um inteiro */

   printf( "Digite o segundo inteiro\n" ); /* prompt */
   scanf( "%d", &inteiro2 ); /* lê um inteiro */

   soma = inteiro1 + inteiro2; /* atribui o total à soma */

   printf( "A soma é %d\n", soma ); /* print soma */

   return 0; /* indica que o programa foi concluído com sucesso */
} /* fim da função main */
```

```
Digite o primeiro inteiro
45
Digite o segundo inteiro
72
A soma é 117
```

Figura 2.5 ■ Programa de adição.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- As linhas 8–10
  - ```
    int integer1; /* primeiro número a ser informado pelo usuário */
    int integer2; /* segundo número a ser informado pelo usuário */
    int sum; /* variável em que a soma será mantida */
    ```
- são definições.
- Os nomes `Inteiro1`, `Inteiro2` e `Soma` são nomes de variáveis.
- Uma variável é um local na memória do computador em
- que um valor pode ser armazenado para ser usado por um programa.
- Essa declaração especifica que as variáveis `inteiro1`, `inteiro2` e `soma` são do tipo `int`, o que significa que guardarão valores inteiros, isto é, números inteiros tais como 7, –11, 0, 31914.
    ```

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- Antes de poderem ser usadas, todas as variáveis devem ser declaradas por um nome e um tipo de dado imediatamente após a chave à esquerda que inicia o corpo de `main`.
- As declarações anteriores poderiam ter sido
- combinadas em uma única instrução de declaração, da seguinte forma:
  - `int inteiro1, inteiro2, soma;`
- mas isso dificultaria a descrição das variáveis nos comentários correspondentes, como fizemos nas linhas de 8 a 10.
- Um nome de variável em C é qualquer identificador válido.
- Um identificador consiste em uma série de caracteres composta por letras, dígitos e o caractere sublinhado (_) que não começa por um dígito.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- Um identificador pode ter qualquer comprimento, mas apenas os 31 primeiros caracteres precisam ser reconhecidos pelos compiladores C, conforme o padrão C.
- C é sensível a maiúsculas e minúsculas, isto é, elas são diferentes em C; assim, `a1` e `A1` são identificadores diferentes.

Erro comum de programação 2.4
Usar uma letra maiúscula onde deveria ter sido usada uma minúscula (por exemplo, digitar Main *no lugar de* main*).*

Dica de prevenção de erro 2.1
Evite identificadores que comecem por sublinhado (_) simples ou duplo, porque aqueles gerados pelo compilador e os da biblioteca-padrão podem usar nomes semelhantes.

Dica de portabilidade 2.1
Use identificadores compostos por, no máximo, 31 caracteres. Isso ajuda a garantir a portabilidade e pode evitar alguns erros de programação sutis.

Boa prática de programação 2.5
A escolha de nomes significativos de variáveis favorecerá a elaboração de um programa 'autodocumentado', isto é, que necessitará de menos comentários.

Boa prática de programação 2.6

A primeira letra de um identificador usado como um nome de variável simples deverá ser uma letra minúscula.
Adiante, atribuiremos um significado especial aos identificadores que começam com letra maiúscula e aos que usam somente letras maiúsculas.

Boa prática de programação 2.7

Nomes de variável com várias palavras podem ajudar a tornar um programa mais legível. Evite juntar as palavras como em comissõestotais*. Em vez disso, separe as palavras com sublinhado, como em* comissões_totais*, ou, se preferir juntar as palavras, inicie cada uma, a partir da segunda, com uma letra maiúscula, como em* comissõesTotais*. O segundo estilo é o melhor.*

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- As declarações precisam ser colocadas após a chave à esquerda de uma função e antes de *quaisquer instruções executáveis*.
- O erro de sintaxe ocorre quando o compilador não consegue reconhecer uma instrução.
- O compilador normalmente emite uma mensagem de erro para ajudá-lo a localizar e a consertar a instrução incorreta.
- Os erros de sintaxe são violações da linguagem.
- Os erros de sintaxe também são chamados erros de compilação ou erros no tempo de compilação.

Erro comum de programação 2.5
A inclusão de declarações de variáveis entre as instruções executáveis causa erros de sintaxe.

Boa prática de programação 2.8
Separe as declarações e as instruções executáveis em uma função com uma linha em branco, para enfatizar onde as definições terminam e as instruções executáveis começam.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- A linha 12
  - `printf( "Digite o primeiro inteiro\n" ); /* prompt */`
- imprime a literal `Digite o primeiro inteiro` na tela e posiciona o cursor no início da próxima linha.
- Essa mensagem é um prompt porque ela pede ao usuário que tome uma atitude específica.
- A instrução seguinte
  - `scanf( "%d", &inteiro1); /* lê um inteiro*/`
- usa `scanf` para obter um valor do usuário..
- A função `scanf` lê o dado de entrada-padrão, que normalmente é o teclado.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- Esse `scanf` tem dois argumentos, `"%d"` e `&inteiro1`.
- O primeiro argumento, a string de controle de formato, indica o tipo de dado que deve ser digitado pelo usuário.
- O especificador de conversão %d indica que os dados devem ser um inteiro (a letra d significa 'inteiro decimal').
- O %, nesse contexto, é tratado pelo `scanf` (e pelo `printf` como veremos adiante) como um caractere especial, que inicia um especificador de conversão.
- O segundo argumento de `scanf` começa com um (&) — chamado de operador de endereço em C —, seguido pelo nome da variável.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- O (&), quando combinado com o nome da variável, informa à `scanf` o local (ou endereço) na memória em que a variável `inteiro1` está armazenada.
- O computador, então, armazena o valor de `inteiro1` nesse local.
- O uso de (&) normalmente confunde os programadores iniciantes e pessoas que tenham programado em outras linguagens que não exigem essa notação.
- Por enquanto, basta que você se lembre de iniciar cada variável em todas as chamadas à `scanf` com o símbolo (&).

Boa prática de programação 2.9
Inclua um espaço após cada vírgula, para que os programas fiquem mais legíveis.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- Quando o computador executa o `scanf`, ele espera o usuário digitar um valor para a variável `inteiro1`.
- O usuário responde digitando um inteiro e apertando a tecla *Enter* para enviar o número ao computador.
- O computador, então, atribui esse número (ou valor)
- à variável `inteiro1`.
- Quaisquer referências posteriores a `inteiro1` nesse programa usarão esse mesmo valor.
- As funções `printf` e `scanf` facilitam a interação entre o usuário e o computador.
- Como essa interação se assemelha a um diálogo, ela é frequentemente chamada de computação conversacional ou computação interativa.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- A linha 15
  - `printf( "Digite o segundo inteiro\n" ); /* prompt */`
- exibe a mensagem Digite o segundo inteiro na tela, e depois posiciona o cursor no início da próxima linha.
- Esse `printf` também solicita ao usuário que ele execute uma ação.
- A instrução
- `scanf( "%d", &integer2 ); /* lê um inteiro*/`
- obtém um valor fornecido pelo usuário para a variável `inteiro2`.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- A instrução de atribuição na linha 18:
  - `soma = inteiro1 + inteiro2; /* atribui total à soma */`
- calcula a soma das variáveis `inteiro1` e `inteiro2` e atribui o resultado à variável soma usando o operador de atribuição =.
- A instrução é lida como "`soma` *recebe* o valor de `inteiro1 + inteiro2`". A maioria dos cálculos é executada em instruções de atribuição.
- O operador = e o operador + são chamados de operadores binários porque cada um tem dois operandos.
- Os dois operandos do operador + são `inteiro1` e `inteiro2`.
- Os dois operandos do operador = são `soma` e o valor da expressão `inteiro1 + inteiro2`.

Boa prática de programação 2.10
Insira espaços dos dois lados de um operador binário. Isso faz com que o operador se destaque, tornando o programa mais legível.

Erro comum de programação 2.6
Um cálculo em uma instrução de atribuição deve ficar do lado direito do operador =. É um erro de compilação colocar um cálculo no lado esquerdo de um operador de atribuição.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- A linha 20
  - `printf( "A soma é %d\n", soma ); /* print soma */`
- chama a função `printf` para imprimir a literal. A `Soma` é seguida pelo valor numérico da variável `soma`.
- Esse `printf` tem dois argumentos, "`Soma é %d\n`" e `soma`.
- O primeiro argumento é a string de controle de formato.
- Ela contém alguns caracteres literais a serem exibidos e o especificador de conversão %d, indicando que um inteiro será exibido.
- O segundo argumento especifica o valor a ser exibido.
- Observe que o especificador de conversão para um inteiro é o mesmo em `printf` e `scanf`.

2.3 Outro programa em C simples: somando dois inteiros (Cont.)

- Poderíamos ter combinado as duas instruções anteriores na instrução:
  - `printf( "A soma é %d\n", inteiro1 + inteiro2 );`
- A linha 22
  - `return 0; /* indica que o programa foi concluído com sucesso */`
- passa o valor 0 para o ambiente do sistema operacional no qual o programa está sendo executado.
- Esse valor indica ao sistema operacional que o programa foi executado com sucesso.
- Para obter mais informações sobre como relatar uma falha do programa, consulte os manuais do ambiente específico de seu sistema operacional.
- A chave à direita, }, na linha 24, indica que o final da função `main` foi alcançado.

Erro comum de programação 2.7
Esquecer de uma ou de ambas as aspas para a string de controle de formato em um printf *ou* scanf*.*

Erro comum de programação 2.8
Esquecer do % em uma especificação de conversão na string de controle de formato de um printf *ou* scanf.

Erro comum de programação 2.9
Colocar uma sequência de escape como \n *fora da string de controle de formato de um* printf *ou* scanf*.*

Erro comum de programação 2.10
Esquecer de incluir as expressões cujos valores devem ser impressos pelo printf *contendo especificadores de conversão.*

Erro comum de programação 2.11
Não fornecer um especificador de conversão quando isso for necessário, em uma string de controle de formato de printf*, para exibir o valor de uma expressão.*

Erro comum de programação 2.12
Colocar dentro da string de controle de formato a vírgula que deveria separar essa string das expressões a serem exibidas.

Erro comum de programação 2.13
Usar o especificador de conversão de formato incorreto ao ler dados com scanf.

Erro comum de programação 2.14
Esquecer de colocar (&) antes de uma variável em uma instrução scanf *quando isso deveria ocorrer.*

Erro comum de programação 2.15
Colocar (&) antes de uma variável incluída em uma instrução printf *quando, na verdade, isso não deveria ocorrer*

2.4 Conceitos de memória

- Nomes de variáveis, tais como `inteiro1`, `inteiro2` e `soma` correspondem, na realidade, a posições na memória do computador.
- Toda variável tem um nome, um tipo e um valor.
- No programa de adição da Figura 2.5, quando a instrução (linha 13):
  - `scanf( "%d", &inteiro1); /* lê um inteiro */`
- executada, o valor digitado pelo usuário é colocado em uma posição de memória à qual o nome `inteiro1` foi designado.
- Suponha que o
- usuário forneça o número `45` como o valor de `inteiro1`.
- O computador colocará `45` na posição `inteiro1` como mostrado na Figura 2.6.

Figura 2.6 ■ Posição de memória que mostra o nome e o valor de uma variável.

2.4 Conceitos de memória (Cont.)

- Sempre que um valor é colocado em uma posição de memória, ele substitui o valor que ocupava esse local anteriormente; assim, a colocação de um novo valor em uma posição da memória é chamada de operação destrutiva.
- Retornando ao nosso programa de adição, quando a instrução (linha 16):
  - `scanf( "%d", &inteiro2); /* lê um inteiro */`
- é executada, suponha que o usuário digite o valor `72`.
- Esse valor é colocado na posição `inteiro2`, e a memória se apresenta como mostra a Figura. 2.7.
- Estas não são, necessariamente, posições adjacentes na memória.

2.4 Conceitos de memória (Cont.)

- Uma vez que o programa tenha obtido os valores para `inteiro1` e `inteiro2`, ele soma esses valores e coloca o resultado da soma na variável `soma`.
- A instrução (linha 18)
  - `Soma = inteiro1 + inteiro2; /* atribui o total à soma */`
- que executa a adição também substitui qualquer valor que esteja armazenado em `soma`.

2.4 Conceitos de memória (Cont.)

- Isso acontece quando a soma calculada de `inteiro1` e `inteiro2` é colocada na posição soma (destruindo o valor já existente em soma).
- Depois que soma é calculada, a memória se apresenta como mostra a Figura 2.8.
- Os valores de `inteiro1` e `inteiro2` são exatamente os que eram antes de serem usados no cálculo.

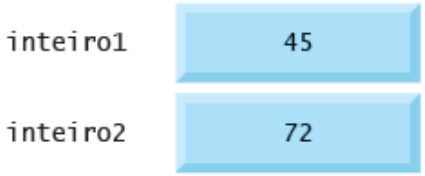

Figura 2.7 ■ Posições de memória depois de os valores para as duas variáveis terem sido fornecidos como entrada.

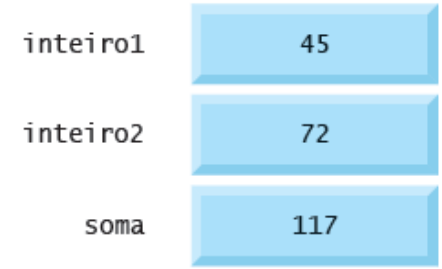

Figura 2.8 ■ Posições de memória após um cálculo.

2.4 Conceitos de memória (Cont.)

- Esses valores foram usados, mas não destruídos, quando o computador executou o cálculo.
- Desse modo, quando um valor é lido de uma posição de memória, o processo é chamado de não destrutivo.

2.5 Aritmética em C

- Os operadores aritméticos em C estão resumidos na Figura 2.9.
- Note o uso de vários símbolos especiais que não são usados em álgebra.
- O asterisco (*) indica multiplicação, e o sinal de porcentagem (%) é o operador-módulo que será discutido adiante.
- Em álgebra, se quisermos multiplicar *a* por *b*, podemos simplesmente colocar esses nomes de variáveis expressos em uma única letra lado a lado, como em *ab*.
- Porém, se fizéssemos isso em C, `ab` seria interpretado como um único nome (ou identificador) de duas letras.
- Portanto, C (e outras linguagens de programação, em geral) exige que a multiplicação seja indicada explicitamente com o uso do operador *, como em a * b.

Operação em C	Operador aritmético	Expressão algébrica	Expressão em C
Adição	+	$f + 7$	f + 7
Subtração	–	$p - c$	p – c
Multiplicação	*	bm	b * m
Divisão	/	x / y ou $\frac{x}{y}$ ou $x \div y$	x / y
Módulo ou resto da divisão entre 2 inteiros	%	$r \bmod s$	r % s

Figura 2.9 ■ Operadores aritméticos.

2.5 Aritmética em C (Cont.)

- Todos os operadores aritméticos são operadores binários.
- Por exemplo, a expressão 3 + 7 contém o operador binário + e os operandos 3 e 7.
- A divisão inteira gera um resultado inteiro.
- Por exemplo, a expressão 7 / 4 é avaliada como 1, e a expressão 17 / 5 é avaliada como 3.
- C oferece o operador de módulo, %, que gera o resto após a divisão inteira.
- A expressão x % y produz o resto depois de x ser dividido por y.
- Assim, 7 % 4 resulta em 3 e 17 % 5 resulta em 2.

Erro comum de programação 2.16

A tentativa de dividir por zero normalmente é indefinida em sistemas de computador, e em geral resulta em um erro fatal, ou seja, um erro que faz com que o programa seja encerrado imediatamente sem ter realizado sua tarefa com sucesso. Os erros não fatais permitem que os programas sejam executados até o fim e, normalmente, produzem resultados incorretos.

2.5 Aritmética em C (Cont.)

- As expressões aritméticas em C precisam ser escritas no formato em linha para facilitar a entrada de programas no computador.
- Assim, expressões como ‘a dividido por b’ devem ser escritas como a/b, para que todos os operadores e operandos apareçam em uma linha reta.
- Geralmente, a notação algébrica

  $\frac{a}{b}$
- não é aceita pelos compiladores, embora alguns pacotes de software de uso especial admitam uma notação mais natural para expressões matemáticas complexas.

2.5 Aritmética em C (Cont.)

- Os parênteses são usados em expressões em C da mesma maneira que em expressões algébricas.
- Por exemplo, para multiplicar `a` pela quantidade `b + c`, escrevemos `a * ( b + c )`.

2.5 Aritmética em C (Cont.)

- C aplica os operadores nas expressões aritméticas em uma sequência precisa, determinada pelas seguintes regras de precedência de operadores, que geralmente são iguais às regras da álgebra:
  - Operadores em expressões contidas dentro de pares de parênteses são calculados primeiro. Assim, os parênteses podem ser usados para forçar a ordem de cálculo a acontecer em uma sequência desejada qualquer. Dizemos que os parênteses estão no 'nível mais alto de precedência'. Em casos de parênteses aninhados, ou embutidos, como
    - `( ( a + b ) + c )`
  - os operadores no par mais interno de parênteses são aplicados primeiro.

2.5 Aritmética em C (Cont.)

- As operações de multiplicação, divisão e módulo são aplicadas primeiro. Se uma expressão contém várias operações de multiplicação, divisão e módulo, a avaliação prossegue da esquerda para a direita. Dizemos que as três operações estão no mesmo nível de precedência.
- As operações de adição e de subtração são avaliadas em seguida. Se uma expressão contém várias operações de adição e subtração, a avaliação prossegue da esquerda para a direita. Ambas as operações têm o mesmo nível de precedência, que é menor do que a precedência das operações de multiplicação, divisão e módulo.

2.5 Aritmética em C (Cont.)

- As regras de precedência de operadores especificam a ordem que C utiliza para avaliar expressões. Quando dizemos que a avaliação prossegue da esquerda para a direita, estamos nos referindo à associatividade dos operadores.
- Veremos que alguns operadores se associam da direita para a esquerda.
- A Figura 2.10 resume essas regras de precedência de operadores.

Operador(es)	Operação(ões)	Ordem de avaliação (precedência)
( )	Parênteses	Avaliados em primeiro lugar. Se os parênteses forem aninhados, a expressão no par mais interno é a primeira a ser avaliada. Se houver vários pares de parênteses 'no mesmo nível' (ou seja, não aninhados), eles serão avaliados da esquerda para a direita.
* / %	Multiplicação Divisão Módulo	Avaliados em segundo lugar. Se houver vários, serão avaliados da esquerda para a direita.
+ –	Adição Subtração	Avaliados por último. Se houver vários, serão avaliados da esquerda para a direita.

Figura 2.10 ■ Precedência de operadores aritméticos.

2.5 Aritmética em C (Cont.)

- A Figura 2.11 ilustra a ordem em que os operadores são aplicados.

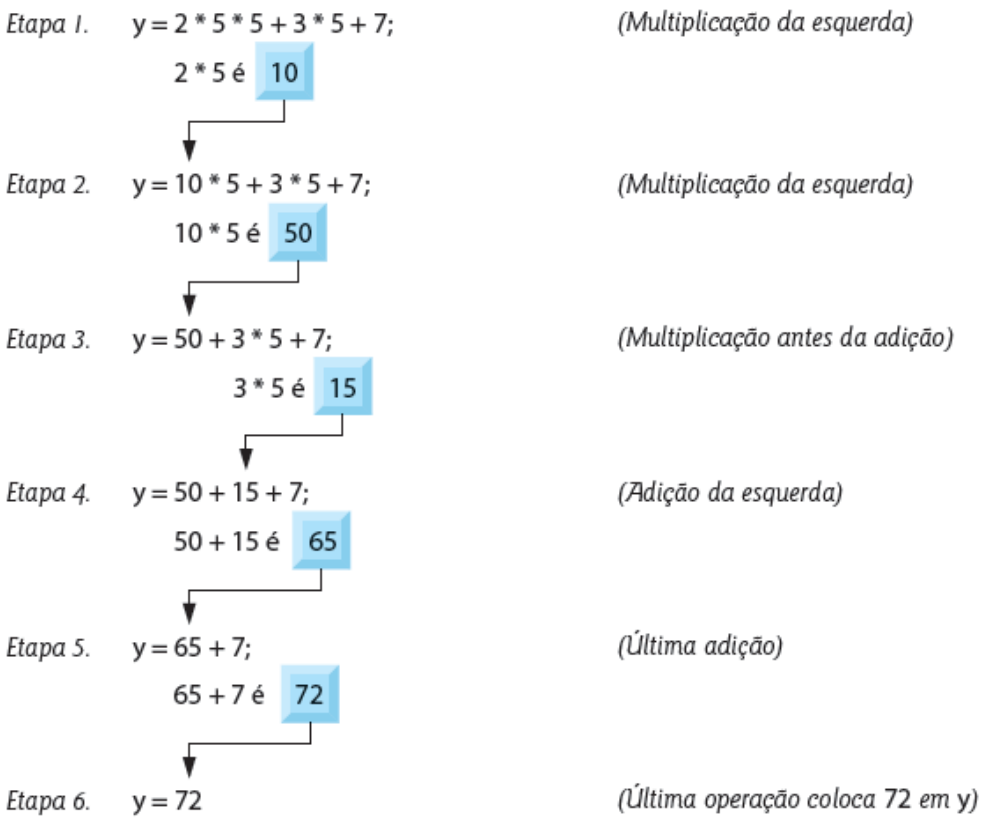

Figura 2.11 ■ Ordem em que um polinômio de segundo grau é avaliado.

2.5 Aritmética em C (Cont.)

- Assim como na álgebra, é aceitável colocar parênteses desnecessários em uma expressão para que ela fique mais clara.
- Estes são chamados de parênteses redundantes.

Boa prática de programação 2.11
O uso de parênteses redundantes em expressões aritméticas mais complexas pode tornar as expressões mais claras.

2.6 Tomada de decisões: operadores relacionais e de igualdade

- As instruções executáveis em C tanto realizam ações (como cálculos ou entrada/saída de dados) quanto tomam decisões (em breve, veremos alguns exemplos disso).
- Por exemplo, em um programa, poderíamos tomar a decisão de determinar se a nota de uma pessoa em uma prova seria maior ou igual a 60, e, se fosse, imprimir a mensagem ‘Parabéns! Você foi aprovado’.
- Essa seção introduz uma versão simples da estrutura if (ou instrução if), a qual permite que um programa tome decisões com base na veracidade ou falsidade de uma condição.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- Se a condição é satisfeita (ou seja, se ela for true — verdadeira), a instrução no corpo da estrutura `if` é executada.
- Se a condição não for satisfeita (ou seja, se for false — falsa), a instrução do corpo não é executada.
- Não importa se o corpo da estrutura é executado ou não; depois que a estrutura `if` termina, a execução prossegue com a próxima instrução após a estrutura `if`.
- As condições em estruturas `if` ser definidas com o uso dos operadores de igualdade e dos operadores relacionais, resumidos na Figura 2.12.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- Os operadores relacionais têm o mesmo nível de precedência, e são associados da esquerda para a direita.
- Os dois operadores de igualdade têm o mesmo nível de precedência, mais baixo que o nível de precedência dos relacionais, e são associados da esquerda para a direita.
- Em C, uma condição pode ser, realmente, qualquer expressão que resulte em um valor zero (falsa) ou diferente de zero (verdadeira).

Operador de igualdade ou relacional na álgebra	Operador de igualdade ou relacional em C	Exemplo de condição em C	Significado da condição em C
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x não é igual a y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior ou igual a y
≤	<=	x <= y	x é menor ou igual a y

Figura 2.12 ■ Operadores de igualdade e relacionais.

Erro comum de programação 2.17
Ocorrerá um erro de sintaxe se os dois símbolos em um dos operadores ==, !=, >= e <= forem separados por espaços.

Erro comum de programação 2.18
Ocorrerá um erro de sintaxe se os dois símbolos em um dos operadores !=, >= e <= forem invertidos, como em =!, => e =<, respectivamente.

Erro comum de programação 2.19
Confundir o operador de igualdade == com o operador de atribuição.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- Para evitar essa confusão, o operador de igualdade deve ser lido como ‘2 iguais’, e o operador de atribuição deve ser lido como ‘recebe’ ou ‘recebe o valor de’.
- Como veremos em breve, confundir esses operadores não causará, necessariamente, um erro de compilação fácil de ser reconhecido, mas poderá causar erros lógicos extremamente sutis.

Erro comum de programação 2.20
Colocar ponto e vírgula imediatamente à direita do parêntese à direita após a condição em uma estrutura if.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- A Figura 2.13 usa seis instruções `if` para comparar dois números informados pelo usuário.
- Se a condição em qualquer uma dessas instruções `if` for verdadeira, então a instrução `printf` associada a esse `if` será executada.

```c
/* Figura 2.13: fig02_13.c
   Usando instruções if, operadores relacionais
   e operadores de igualdade */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   int num1; /* primeiro número do usuário a ser lido */
   int num2; /* segundo número do usuário a ser lido */

   printf( "Entre com dois inteiros e eu lhe direi\n" );
   printf( "as relações que eles satisfazem: " );

   scanf( "%d%d", &num1, &num2 ); /* lê dois inteiros */

   if ( num1 == num2 ) {
      printf( "%d é igual a %d\n", num1, num2 );
   } /* fim do if */

   if ( num1 != num2 ) {
      printf( "%d não é igual a %d\n", num1, num2 );
   } /* fim do if */

   if ( num1 < num2 ) {
      printf( "%d é menor que %d\n", num1, num2 );
   } /* fim do if */

   if ( num1 > num2 ) {
      printf( "%d é maior que %d\n", num1, num2 );
   } /* fim do if */

   if ( num1 <= num2 ) {
      printf( "%d é menor ou igual a %d\n", num1, num2 );
   } /* fim do if */

   if ( num1 >= num2 ) {
      printf( "%d é maior ou igual a %d\n", num1, num2 );
   } /* fim do if */

   return 0; /* indica que o programa foi concluído com sucesso */
} /* fim da função main */
```

```
Entre com dois inteiros e eu lhe direi
as relações que eles satisfazem: 3 7
3 não é igual a 7
3 é menor que 7
3 é menor ou igual a 7
```

```
Entre com dois inteiros e eu lhe direi
as relações que eles satisfazem: 22 12
22 não é igual a 12
22 é maior que 12
22 é maior ou igual a 12
```

```
Entre com dois inteiros e eu lhe direi
as relações que eles satisfazem: 7 7
7 é igual a 7
7 é menor ou igual a 7
7 é maior ou igual a 7
```

Figura 2.13 ■ Usando instruções `if`, operadores relacionais e operadores de igualdade.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- O programa utiliza a `scanf` (line 15) (linha 15) para inserir dois números.
- Cada especificador de conversão tem um argumento correspondente em que um valor será armazenado.
- O primeiro `%d` converte um valor a ser armazenado na variável `num1`, e o segundo `%d` converte um valor a ser armazenado na variável `num2`.
- O recuo do corpo de cada instrução `if` e a inclusão de linhas em branco acima e abaixo de cada estrutura melhora a legibilidade do programa.

Boa prática de programação 2.12
Recue a(s) instrução(ões) no corpo de uma estrutura if*.*

Boa prática de programação 2.13
Insira uma linha em branco antes e depois de cada estrutura if *em um programa, por questão de legibilidade.*

Boa prática de programação 2.14
Embora seja permitido, não deve haver mais de uma instrução por linha em um programa.

Erro comum de programação 2.21
Incluir vírgulas (quando nenhuma é necessária) entre os especificadores de conversão na string de controle de formato de uma instrução scanf.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- Uma chave à esquerda, {, inicia o corpo de cada estrutura `if` (por exemplo, na linha 17).
- Uma chave à direita correspondente, }, encerra o corpo de cada estrutura `if` (por exemplo, na linha 19).
- Qualquer quantidade de instruções pode ser colocada no corpo de uma estrutura `if`.
- Ocomentário (linhas 1-3) na Figura 2.13 é dividido em três linhas.
- Em programas em C, os caracteres de espaço em branco como tabulações, novas linhas e espaços normalmente são ignorados.
- Assim, instruções e comentários podem ser divididos em várias linhas.
- Porém, não é correto separar identificadores.

Boa prática de programação 2.15
Uma instrução extensa pode se espalhar por várias linhas. Se uma instrução for dividida em várias linhas, escolha pontos de quebra que façam sentido (por exemplo, após uma vírgula em uma lista separada por vírgulas). Se uma instrução for dividida por duas ou mais linhas, recue todas as linhas seguintes.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- A Figura 2.14 lista a precedência dos operadores apresentados neste capítulo.
- Os operadores aparecem de cima para baixo em ordem decrescente de precedência.
- O sinal de igual também é um operador.
- Todos esses operadores, com exceção do operador de atribuição =, são associados da esquerda para a direita.
- O operador de atribuição (=) avalia da direita para a esquerda.

Boa prática de programação 2.16
Consulte o quadro de precedência de operadores ao escrever expressões contendo muitos operadores. Confirme se os operadores estão empregados em ordem correta na expressão. Se você não tiver certeza da ordem de avaliação em uma expressão complexa, use parênteses para agrupar as expressões ou quebre a instrução em várias instruções mais simples. Não se esqueça de observar que alguns dos operadores em C, como o operador de atribuição (=), são associados da direita para a esquerda, e não da esquerda para a direita.

Operadores				Associatividade
()				esquerda para direita
*	/	%		esquerda para direita
+	–			esquerda para direita
<	<=	>	>=	esquerda para direita
==	!=			esquerda para direita
=				direita para esquerda

Figura 2.14 ■ Precedência e associatividade dos operadores discutidos até aqui.

2.6 Tomada de decisões: operadores relacionais e de igualdade (Cont.)

- Neste capítulo, algumas das palavras que usamos nos programas em C — em particular `int`, `return` e `if` são palavras-chave, ou palavras reservadas da linguagem..
- A Figura 2.15 contém as palavras-chave em C.
- Essas palavras têm significado especial para o compilador C, de modo que você precisa ter cuidado para não usá-las como identificadores, como em nomes de variáveis.

Palavras-chave			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while
Palavras-chave acrescentadas na C99			
_Bool _Complex _Imaginary inline restrict			

Figura 2.15 ■ Palavras-chave em C.