Desenvolvimento Estruturado de Programas em C

C Como Programas, Sexta Edição

Objetivos

Neste capítulo, você aprenderá:

- Técnicas básicas para a solução de problemas.
- A desenvolver algoritmos por meio do processo de refinamento top-down, com melhorias sucessivas.
- A utilizar a estrutura de seleção if e a estrutura de seleção if…else para selecionar ações.
- A usar a estrutura de repetição while para executar instruções em um programa repetidamente.
- Repetição controlada por contador e repetição controlada por sentinela.
- Programação estruturada.
- Operadores de incremento, decremento e atribuição.

3.1 Introdução

- Antes de escrever um programa a fim de resolver um problema em particular, é essencial que se tenha uma compreensão plena e profunda desse problema, e que se faça uso de uma abordagem cuidadosamente planejada para resolvê-lo.
- Os dois capítulos seguintes discutem técnicas que facilitam o desenvolvimento de programas de computador estruturados.

3.2 Algoritmos

- Podemos resolver qualquer problema de computação ao executarmos uma série de ações em uma ordem específica.
- Um dos procedimentos utilizados para resolver um problema em termos
  - das ações a serem executadas e
  - da ordem em que essas ações devem ser executadas
- é chamado de algoritmo.
- É importante especificar corretamente a ordem em que as ações serão executadas

3.2 Algoritmos (Cont.)

- Considere o algoritmo ‘preparar-se para ir trabalhar’, seguido de um executivo júnior saindo da cama e indo para o trabalho: (1) sair da cama, (2) tirar o pijama, (3) tomar banho, (4) vestir-se, (5) tomar o café da manhã e (6) dirigir até o trabalho.
- Essa rotina faz com que o executivo chegue ao trabalho bem preparado para tomar decisões críticas.

3.2 Algoritmos (Cont.)

- Suponha que os mesmos passos sejam executados em uma ordem ligeiramente diferente: (1) sair da cama, (2) tirar o pijama, (3) vestir-se, (4) tomar banho, (5) tomar o café da manhã e (6) dirigir até o trabalho.
- Nesse caso, nosso executivo se apresentaria para trabalhar literalmente ensopado.
- Especificar a ordem em que os comandos devem ser executados em um programa de computador é chamado de controle do programa.

3.3 Pseudocódigo

- O pseudocódigo é uma linguagem artificial e informal que ajuda os programadores a desenvolver algoritmos.
- Ele é semelhante à linguagem do dia a dia; é conveniente e fácil de usar, embora não seja realmente uma linguagem de programação para computadores.
- Os programas em pseudocódigo não são executados em computadores.
- Em vez disso, ele ajuda o programador a ‘conceber’ um programa antes de tentar escrevê-lo em uma linguagem de programação, tal como C..
- O pseudocódigo consiste puramente em caracteres, de modo que você pode escrever programas em pseudocódigo convenientemente usando apenas um programa editor.

3.3 Pseudocódigo (Cont.)

- Um programa em pseudocódigo cuidadosamente preparado pode ser convertido facilmente em um programa correspondente em C.
- O pseudocódigo consiste somente em comandos executáveis — aqueles que são executados quando o programa é convertido de pseudocódigo para C e, depois, executado em C.
- As definições não são comandos executáveis.
- Elas são mensagens para o compilador.

3.3 Pseudocódigo (Cont.)

- Por exemplo, a definição
  - `int i;`
- simplesmente diz ao compilador o tipo da variável i, e instrui o compilador a reservar espaço na memória para essa variável.
- Mas essa declaração não provoca nenhuma ação — entrada, saída ou cálculo — que deverá ocorrer quando o programa for executado.
- Alguns programadores escolhem listar as variáveis e mencionar brevemente o propósito de cada uma no início de um programa em pseudocódigo.

3.4 Estruturas de controle

- Em um programa, normalmente, os comandos são executados um após do outro, na sequência em que estiverem escritos.
- Isso é chamado de execução sequencial..
- Vários comandos em C, que discutiremos em breve, permitem ao programador especificar que o próximo comando a ser executado pode ser outro que não o próximo na sequência.
- Isso é chamado de transferência de controle.
- Durante a década de 1960, ficou claro que o uso indiscriminado de transferências de controle era a raiz de muitas das dificuldades experimentadas por grupos de desenvolvimento de software.

3.4 Estruturas de controle (Cont.)

- O comando `goto` foi considerado culpado, porque permite ao programador especificar uma transferência de controle para uma variedade muito grande de destinos possíveis em um programa.
- A noção da chamada programação estruturada tornou-se quase sinônimo de 'eliminação de `goto`'
- Pesquisas demonstraram que os programas podiam ser escritos sem quaisquer comandos `goto`.
- O desafio para os programadores daquela época era mudar o estilo de programação: 'escrever programas sem usar o comando `goto`'.

3.4 Estruturas de controle (Cont.)

- Os resultados foram impressionantes, como perceberam grupos de desenvolvimento de software: houve reduções no tempo de desenvolvimento, os sistemas passaram a ser entregues dentro do prazo e os projetos de software começaram a ser concluídos dentro do orçamento com mais frequência.
- Os programas produzidos com técnicas estruturadas eram mais claros, fáceis de depurar e modificar, e a probabilidade de serem isentos de erros desde o início era maior.
- Pesquisa demonstrou que todos os programas podiam ser escritos nos termos de somente três estruturas de controle: estrutura de sequência, estrutura de seleção e estrutura de repetição.

3.4 Estruturas de controle (Cont.)

- A sequência está embutida na linguagem em C.
- A menos que seja instruído de outra forma, o computador executará os comandos de C um após do outro, na sequência em que eles estiverem escritos.
- O segmento de fluxograma da Figura 3.1 mostra uma estrutura de sequência típica em C.
- Um fluxograma é uma representação gráfica de um algoritmo, ou de uma parte de um algoritmo.
- Fluxogramas são desenhados a partir de certos símbolos especiais, tais como retângulos, losangos, elipses e pequenos círculos; esses símbolos são conectados por setas chamadas linhas de fluxo.

3.4 Estruturas de controle (Cont.)

- Assim como o pseudocódigo, os fluxogramas são úteis no desenvolvimento e na representação de algoritmos, embora o pseudocódigo seja o preferido da maioria dos programadores.
- Considere o fluxograma para a estrutura de sequência na Figura 3.1.
- Usamos o retângulo, também chamado de símbolo de ação, para indicar qualquer tipo de ação, inclusive um cálculo ou uma operação de entrada/saída.
- As linhas de fluxo na figura indicam a sequência em que as ações deverão ser executadas — em primeiro lugar, `nota` deve ser somado a `total`, e depois `1` deve ser somado a `contador`.
- C permite que, em uma estrutura de sequência, tenhamos tantas ações quantas quisermos.

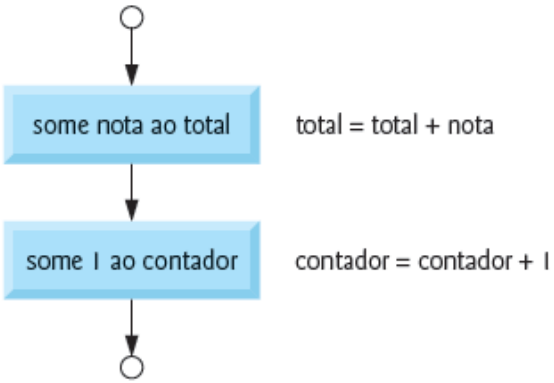

Figura 3.1 ■ Colocando a estrutura de sequência em C em um fluxograma.

3.4 Estruturas de controle (Cont.)

- Ao desenhar um fluxograma que represente um algoritmo completo, uma elipse contendo a palavra ‘Início’ será o primeiro símbolo a ser usado; uma elipse contendo a palavra ‘Fim’ será o último símbolo a ser utilizado.
- Ao desenhar somente uma parte de um algoritmo,
- como na Figura 3.1, as elipses serão omitidas; em seu lugar, usaremos pequenos círculos, também chamados de conectores.
- Talvez o mais importante símbolo na elaboração de fluxogramas seja o losango, também chamado de símbolo de decisão, que indica a necessidade de tomar uma decisão.

3.4 Estruturas de controle (Cont.)

- C oferece três tipos de estruturas de seleção em forma de comandos.
- A estrutura de seleção `if` (Seção 3.5) tanto executa (seleciona) uma ação, se uma condição for verdadeira, quanto 'pula' a ação, se a condição for falsa.
- A estrutura de seleção `if...else` (Seção 3.6) executa uma ação, se uma condição for verdadeira, e executa uma ação diferente, se a condição for falsa.
- O comando de seleção `switch` (a ser discutido no Capítulo 4) executa uma das muitas ações diferentes que dependem do valor de uma expressão.

3.4 Estruturas de controle (Cont.)

- O comando `if` é chamado de comando de seleção única, pois seleciona ou ignora uma única ação.
- O comando `if...else` um comando de seleção dupla, pois seleciona uma dentre duas ações diferentes.
- O comando `switch` é chamado de comando de seleção múltipla, pois seleciona a ação a ser executada dentre muitas ações diferentes.
- C fornece três tipos de estruturas de repetição em forma de comandos, a saber: `while` (Seção 3.7), `do...while` e `for` (esses dois últimos serão abordados no Capítulo 4).
- Bem, isso é tudo!

3.4 Estruturas de controle (Cont.)

- C tem apenas sete estruturas de controle: sequência, três tipos de seleção e três tipos de repetição.
- Cada programa em C é formado pela combinação de muitas estruturas de controle, de acordo com o que for apropriado para o algoritmo que o programa esteja implementando.
- Assim como ocorre com a estrutura de sequência da Figura 3.1, veremos que cada estrutura de controle representada por um fluxograma contém dois círculos pequenos, um no ponto de entrada do comando de controle e um no ponto de saída.
- Esses comandos de controle de entrada e saída únicas facilitam a construção de programas.

3.4 Estruturas de controle (Cont.)

- Os segmentos do fluxograma do comando de controle podem ser ligados um ao outro ao conectarmos o ponto de saída de um comando ao ponto de entrada do seguinte.
- Isso é muito parecido com o modo como as crianças empilham peças de montagem, e, por isso, chamamos esse processo de empilhamento do comando de controle.
- Veremos que só existe outra maneira de conectar os comandos de controle — um método chamado de aninhamento do comando de controle.
- Assim, qualquer programa em C que tenhamos de montar pode ser construído a partir de apenas sete tipos de comandos de controle combinados de duas maneiras.
- Isso é a essência da simplicidade.

3.5 A estrutura de seleção `if`

- As estruturas de seleção são usadas na escolha entre cursos de ação alternativos.
- Por exemplo, suponha que a nota de corte em um exame seja 60.
- O comando em pseudocódigo
  - Se a nota do aluno for maior ou igual a 60
  - Imprima ‘Aprovado’
- determina se a condição ‘nota do aluno é maior ou igual a 60’ é verdadeira ou falsa.
- Se a condição é verdadeira, então a palavra ‘Aprovado’ é impressa, e o próximo comando na sequência do pseudocódigo é ‘executado’ (lembre-se de que o pseudocódigo não é uma autêntica linguagem de programação).

3.5 A estrutura de seleção `if` (Cont.)

- Se a condição é falsa, o comando de impressão é ignorado e o próximo comando na sequência do pseudocódigo é executado.
- Há um recuo na segunda linha dessa estrutura de seleção.
- Tal recuo é opcional, mas altamente recomendado, pois enfatiza a estrutura inerente aos programas estruturados.
- O compilador de C ignora caracteres de espaçamento, como caracteres em branco, pontos de tabulação e caracteres de nova linha, usados para recuo e espaçamento vertical.

Boa prática de programação 3.1

Aplicar consistentemente convenções razoáveis de recuo faz com que a legibilidade do programa seja muito maior. Sugerimos uma marca de tabulação com um tamanho fixo de cerca de 1/4 de polegada, ou três espaços, por nível de recuo.

3.5 A estrutura de seleção `if` (Cont.)

- O comando *Se* do pseudocódigo apresentado pode ser escrito em C como
  - ```
    if ( nota >= 60 ) {
       printf( "Aprovado\n" );
    } /* fim do if */
    ```
- Note que o código em C corresponde de maneira aproximada ao pseudocódigo.
    ```

Boa prática de programação 3.2
Frequentemente, o pseudocódigo é usado na 'criação' de um programa durante o processo de planejamento. Depois disso, o programa em pseudocódigo é convertido em C.

3.5 A estrutura de seleção `if` (Cont.)

- O fluxograma da Figura 3.2 mostra a estrutura `if` de seleção única.
- Ele contém o que talvez seja o símbolo mais importante na elaboração de fluxogramas — o losango, também é conhecido como símbolo de decisão, pois indica que uma decisão deve ser tomada.
- O símbolo de decisão contém uma expressão — uma condição — que pode ser verdadeira ou falsa.

3.5 A estrutura de seleção if (Cont.)

- Duas linhas de fluxo têm sua origem no símbolo de decisão.
- Uma indica a direção a ser seguida quando a expressão dentro do símbolo é verdadeira; a outra indica a direção a ser tomada quando a expressão é falsa.
- As decisões podem ser baseadas em condições que contêm operadores relacionais ou de igualdade.
- Na verdade, uma decisão pode ser tomada com base em qualquer expressão; se o valor da expressão é zero, ela é tratada como falsa, e se o valor da expressão não é zero, ela é tratada como verdadeira.

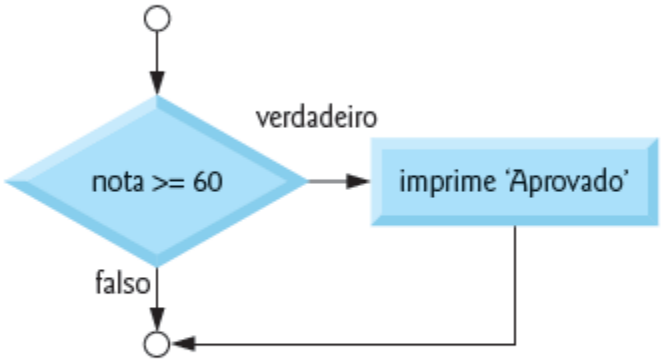

Figura 3.2 ■ Elaboração do fluxograma do comando `if` de seleção única.

3.5 A estrutura de seleção `if` (Cont.)

- A estrutura `if` também é uma estrutura de entrada e saída únicas.
- Podemos imaginar sete caixas, cada uma contendo somente estruturas de controle de um dos sete tipos.
- Esses segmentos de fluxograma estão vazios; nada está escrito nos retângulos ou nos losangos.
- Sua tarefa é montar um programa que junte as peças de cada tipo de estrutura de controle de que o algoritmo necessita, combinando esses comandos de controle dos dois únicos modos possíveis (empilhamento ou aninhamento), e depois preenchendo-os com ações e decisões convenientes ao algoritmo.

3.6 A estrutura de seleção `if...else`

- A estrutura de seleção `if...else` permite que você determine que uma ação deve ser executada quando a condição for verdadeira, e não quando a condição for falsa.
- Por exemplo, o comando de pseudocódigo
  - Se a nota do aluno for maior ou igual a 60
    Imprima 'Aprovado'
    se não
    Imprima 'Reprovado'
- imprime *Aprovado* se a nota do aluno é maior ou igual a 60, e imprime *Reprovado* se a nota do aluno é menor que 60.
- Em ambos os casos, depois de ocorrida a impressão, o próximo comando de pseudocódigo da sequência é 'executado'..

Boa prática de programação 3.3
Recuar ambos os comandos do corpo de uma estrutura if...else*.*

Boa prática de programação 3.4
Se existem vários níveis de recuo, cada nível deve ser recuado pelo mesmo espaço adicional.

3.6 A estrutura de seleção `if...else` (Cont.)

- A estrutura de pseudocódigo *Se...se não* pode ser escrita em C como

```
if ( nota >= 60 ) {
   printf( "Aprovado\n" );
} /* fim do if */
else {
   printf( "Reprovado\n" );
} /* fim do else */
```

- O fluxograma da Figura 3.3 mostra bem o fluxo de controle na estrutura `if...else`.
- Uma vez mais, note que (além de círculos pequenos e setas) os únicos símbolos no fluxograma são retângulos (para ações) e um losango (para uma decisão).

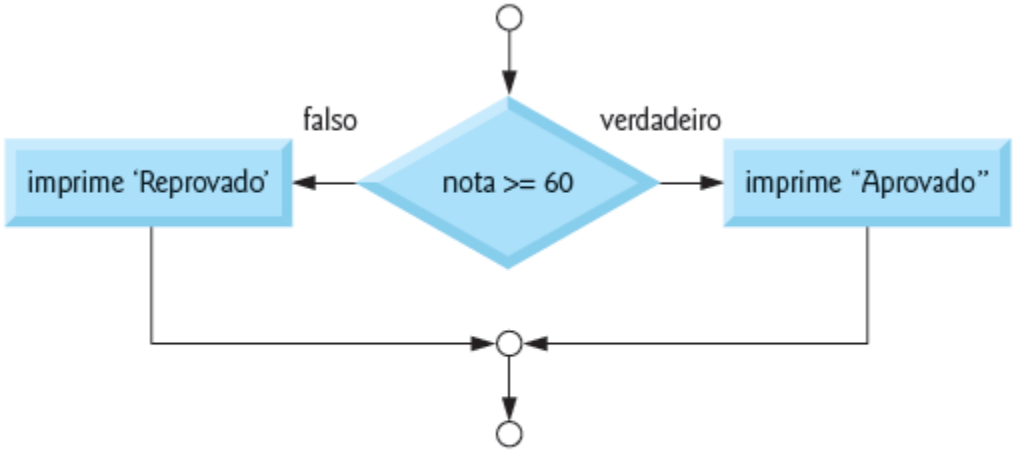

Figura 3.3 ■ Fluxograma da estrutura de seleção dupla `if...else`.

3.6 A estrutura de seleção `if...else` (Cont.)

- C oferece o operador condicional (?:) — ele aceita três operandos.
- Os operandos, com o operador condicional, formam uma expressão condicional.
- O primeiro operando é uma condição.
- O segundo operando é o valor para a expressão condicional inteira, se a condição for verdadeira; e o terceiro operando é o valor para a expressão condicional inteira se a condição for falsa..

3.6 A estrutura de seleção `if...else` (Cont.)

- Por exemplo, o comando `printf`
  - `printf( "%s\n", nota >= 60 ? "Aprovado" : "Reprovado" );`
- contém uma expressão condicional que resulta na string literal "Aprovado" se a condição nota >= 60 for verdadeira, e resulta na string "Reprovado" se a condição for falsa.
- A string de controle de formato do `printf` contém a especificação de conversão %s para imprimir uma string de caracteres.
- Desse modo, esse comando `printf` executa essencialmente o mesmo que o comando `if...else` anterior.

3.6 A estrutura de seleção `if...else` (Cont.)

- O segundo e o terceiro operandos em uma expressão condicional também podem ser ações a serem executadas.
- Por exemplo, a expressão condicional
  - ```
    Nota >= 60 ? printf( "Aprovado\n" ) :
        printf( "Reprovado\n" );
    ```
- é lida como "Se nota é maior ou igual a 60, então `printf("Aprovado\n")`, caso contrário, `printf("Aprovado\n")`." Isso também é comparável à estrutura `if...else` anterior.
- Veremos que os operadores condicionais podem ser usados em algumas situações em que as estruturas `if...else` não podem.
    ```

3.6 A estrutura de seleção `if...else` (Cont.)

- Estruturas `if...else` aninhadas testam vários casos, colocando estruturas `if...else` dentro de estruturas `if...else`.
- Por exemplo, a estrutura de pseudocódigo a seguir imprimirá A para notas de exame maiores ou iguais a 90, B para notas maiores ou iguais a 80, C para notas maiores ou iguais a 70, D para notas maiores ou iguais a 60, e F para todas as outras notas.
  - Se a nota do aluno é maior ou igual a 90
  - Imprime ‘A’
  - se não
  - Se a nota do aluno é maior ou igual a 80
  - Imprime ‘B’
  - se não
  - Se a nota do aluno é maior ou igual a 70
  - Imprime ‘C’
  - se não
  - Se a nota do aluno é maior ou igual a 60
  - Imprime ‘D’
  - se não
  - Imprime ‘F’

3.6 A estrutura de seleção if...else (Cont.)

- Esse pseudocódigo pode ser escrito em C como

```
if ( nota >= 90 )
   printf( "A\n" );
else
   if ( nota >= 80 )
      printf("B\n");
   else
      if ( nota >= 70 )
         printf("C\n");
      else
         if ( nota >= 60 )
            printf( "D\n" );
         else
            printf( "F\n" );
```

3.6 A estrutura de seleção `if`... `else` (Cont.)

- Se a variável `nota` for maior ou igual a 90, as quatro primeiras condições serão verdadeiras, mas somente o `printf` após o primeiro teste será executado.
- Depois que esse `printf` executado, a `else` da extruturas `if`... `else` 'externa' é desprezada.

3.6 A estrutura de seleção `if...else` (Cont.)

- Muitos programadores em C preferem escrever a estrutura `if` anterior como

```
if ( nota >= 90 )
   printf( "A\n" );
else if ( nota >= 80 )
   printf( "B\n" );
else if ( nota >= 70 )
   printf( "C\n" );
else if ( nota >= 60 )
   printf( "D\n" );
else
   printf( "F\n" );
```

3.6 A estrutura de seleção `if...else` (Cont.)

- Para o compilador C, as duas formas são equivalentes.
- A última forma é popular porque evita o recuo profundo do código para a direita.
- A estrutura de seleção `if` espera que exista apenas uma instrução em seu corpo.
- Para incluir várias instruções no corpo de um `if`, delimite o conjunto de instruções com chaves ({ e }).
- Um conjunto de instruções contidas dentro de um par de chaves é chamado de instrução composta ou bloco.

Observação sobre engenharia de software 3.1
Uma instrução composta pode ser colocada em qualquer lugar de um programa em que uma única instrução pode ser colocada.

3.6 A estrutura de seleção `if...else` (Cont.)

- O exemplo a seguir inclui uma instrução composta na parte `else` de uma estrutura `if...else`.

```
if ( nota >= 60 ) {
   printf( “Aprovado.\n” );
} /* fim do if */
else {
   printf( “Reprovado.\n” );
   printf( "Você precisa fazer esse curso
novamente.\n" );
} /* fim do else */
```

3.6 A estrutura de seleção `if...else` (Cont.)

- Nesse caso, se a nota for menor que 60, o programa executará as duas instruções `printf` no corpo do `else` e imprimirá:
  - `Reprovado.`
  - `Você precisa fazer esse curso novamente.`
- Observe as chaves ao redor das duas instruções na cláusula `else`.
- These braces are important. Essas chaves são importantes. Sem elas, a instrução

```
printf( "Você precisa fazer esse curso
novamente.\n" );
```

- ficaria fora do corpo da parte `else` do `if`, e seria executada não importando se a nota fosse menor que 60.

Erro comum de programação 3.1
Esquecer-se de uma ou de ambas as chaves que delimitam uma instrução composta.

3.6 A estrutura de seleção `if...else` (Cont.)

- Um erro de sintaxe é detectado pelo compilador.
- Um erro lógico tem seu efeito durante a execução.
- Um erro lógico fatal faz com que o programa falhe e seja encerrado prematuramente.
- Um erro lógico não fatal permite que o programa continue executando, mas produzindo resultados incorretos.

Observação sobre engenharia de software 3.2
Assim como uma instrução composta pode ser colocada em qualquer lugar em que uma única instrução pode ser colocada, também é possível não haver instrução nenhuma, ou seja, uma instrução vazia. A instrução vazia é representada por um ponto e vírgula (;) no local em que uma instrução normalmente estaria.

Erro comum de programação 3.2
Colocar um ponto e vírgula após a condição de uma estrutura if*, como em (nota >= 60);, ocasiona um erro lógico nas estruturas* if *de seleção única e um erro de sintaxe nas estruturas* if *de seleção dupla.*

Dica de prevenção de erro 3.1
Digitar as chaves de início e de fim das instruções compostas antes de digitar as instruções individuais dentro das chaves ajuda a evitar a omissão de uma ou de ambas as chaves, o que impede erros de sintaxe e erros lógicos (onde as duas chaves realmente forem necessárias).

3.7 A estrutura de repetição while

- Uma estrutura de repetição permite que você especifique que uma ação deverá ser repetida enquanto alguma condição permanecer verdadeira.
- A estrutura em pseudocódigo
  - Enquanto houver mais itens na minha lista de compras
  - Comprar próximo item e riscá-lo da minha lista
- descreve a repetição que ocorre durante uma ida às compras.
- A condição ‘existem mais itens na minha lista de compras’ pode ser verdadeira ou falsa.
- Se for verdadeira, então a ação ‘Comprar próximo item e riscá-lo da minha lista’ será realizada.
- Essa ação será realizada repetidamente enquanto a condição permanecer verdadeira.

3.7 A estrutura de repetição `while` (Cont.)

- A instrução (ou instruções) contida na estrutura de repetição *while* (enquanto) constitui o corpo do *while*.
- O corpo da estrutura *while* pode ser uma única instrução ou uma instrução composta.
- Eventualmente, a condição se tornará falsa (quando o último item tiver sido comprado e riscado da lista).
- Nesse ponto, a repetição terminará e a primeira instrução do pseudocódigo após a estrutura de repetição será executada.

Erro comum de programação 3.3
Não fornecer o corpo de uma estrutura while com uma ação que eventualmente fará com que a condição no while se torne falsa. Normalmente, essa estrutura de repetição nunca termina — esse erro é chamado de 'loop infinito'.

Erro comum de programação 3.4
Escrever a palavra-chave while com um W maiúsculo, como em While (lembre-se de que C é uma linguagem que diferencia maiúsculas de minúsculas). Todas as palavras-chave reservadas de C, como while, if e else, contêm apenas letras minúsculas.

3.7 A estrutura de repetição while (Cont.)

- Como exemplo de um `while` real, consider considere um segmento de programa projetado para encontrar a primeira potência de 3 maior que 100.
- Suponha que a variável inteirae `produto` tenha sido inicializada com 3.
- Quando a seguinte estrutura de repetição `while` acabar de ser executada, `produto` terá a resposta desejada:
  - `produto = 3;`
  - ```
    while ( produto <= 100 ) {
      produto = 3 * produto ;
    } /* fim do while */
    ```
- O fluxograma da Figura 3.4 ilustra bem o fluxo de controle na estrutura de repetição `while`.
    ```

3.7 A estrutura de repetição `while` (Cont.)

- Mais uma vez, observe que (além de pequenos círculos e setas) o fluxograma contém apenas um retângulo e um losango.
- O fluxograma mostra claramente a repetição.
- A linha de fluxo que sai do retângulo volta para a decisão, que é testada toda vez que o loop é repetido, até que a decisão, por fim, torne-se falsa.
- Nesse ponto, a estrutura `while` termina, e o controle passa para a próxima instrução no programa.

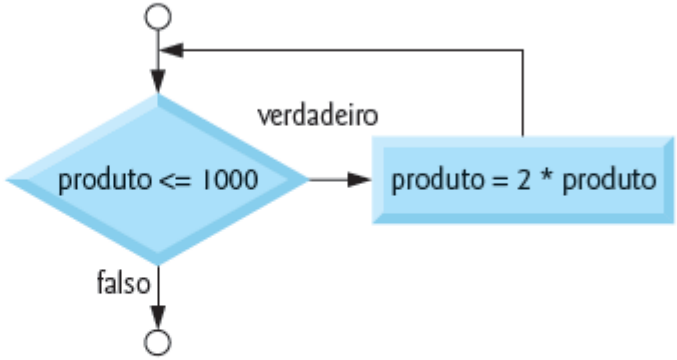

Figura 3.4 ■ Fluxograma da estrutura de repetição `while`.

3.7 A estrutura de repetição `while` (Cont.)

- Quando a estrutura `while` é iniciada, o valor de `produto` é 3.
- A variável `produto` é multiplicada repetidamente por 3, assumindo os valores 9, 27 e 81, sucessivamente.
- Quando `product` chega a 243, a condição na estrutura `while`, `produto <= 100`, torna-se falsa.
- Isso encerra a repetição, e o valor final de `produto` é 243.
- A execução do programa continuará com a instrução seguinte ao `while`.

3.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador)

- Para ilustrar como os algoritmos são desenvolvidos, resolveremos diversas variantes do problema de calcular a média de uma turma.
- Considere o seguinte problema:
  - *Uma turma de dez alunos realiza um teste. As notas (inteiros, no intervalo de 0 a 100) dadas aos alunos estão à sua disposição. Determine a média das notas da turma.*
- A média da turma é igual à soma das notas dividida pelo número de alunos.
- O algoritmo para resolver esse problema em um computador deve inserir cada uma das notas, executar o cálculo da média e imprimir o resultado.

3.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador) (Cont.)

- Usemos o pseudocódigo para listar as ações que deverão ser executadas e especificar a ordem em que essas ações deverão ser executadas.
- Usamos uma repetição controlada por contador para inserir as notas, uma de cada vez.
- Essa técnica usa uma variável chamada contador para especificar o número de vezes que um conjunto de comandos será executado.
- Nesse exemplo, a repetição termina quando o contador exceder 10.

3.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador) (Cont.)

- Nesta seção, apresentamos um algoritmo em pseudocódigo (Figura 3.5) e o programa correspondente em C (Figura 3.6).
- Na próxima seção, mostraremos como algoritmos em pseudocódigo são desenvolvidos.
- A repetição controlada por contador é chamada, frequentemente, de repetição definida, porque o número de repetições é conhecido antes que o loop comece a ser executado.

```
1  Define total como zero
2  Define contador de notas como um
3
4  Enquanto contador de notas é menor ou igual a dez
5      Lê a próxima nota
6      Soma a nota ao total
7      Soma um ao contador de notas
8
9  Define a média da turma como o total dividido por dez
10 Imprime a média da turma
```

Figura 3.5 ■ Algoritmo de pseudocódigo que usa a repetição controlada por contador para resolver o problema da média da turma.


```c
/* Figura 3.6: fig03_06.c
   Programa de média da turma com repetição controlada por contador */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   int contador; /* número da nota a digitar em seguida */
   int nota;     /* valor da nota */
   int total;    /* soma das notas inseridas pelo usuário */
   int média;    /* média das notas */

   /* fase de inicialização */
   total = 0;    /* inicializa total */
   contador = 1; /* inicializa contador do loop */

   /* fase de processamento */
   while ( contador <= 10 ) {      /* loop 10 vezes */
      printf( "Digite a nota: " ); /* prompt para inserção */
      scanf( "%d", &nota );        /* lê a nota do usuário */
      total = total + nota;        /* soma nota ao total */
      contador = contador + 1;     /* incrementa contador */
   } /* fim do while */

   /* fase de término */
   média = total / 10; /* divisão de inteiros */

   printf( "Média da turma é %d\n", média ); /* exibe resultado */
   return 0; /* indica que programa foi concluído com sucesso */
} /* fim da função main */
```

```
Digite a nota: 98
Digite a nota: 76
Digite a nota: 71
Digite a nota: 87
Digite a nota: 83
Digite a nota: 90
Digite a nota: 57
Digite a nota: 79
Digite a nota: 82
Digite a nota: 94
Média da turma é 81
```

Figura 3.6 ■ Programa em C e exemplo de sua execução para o problema de média da turma com repetição controlada por contador.

3.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador) (Cont.)

- Observe as referências no algoritmo à variável total e à variável contador.
- A variável total é usada para acumular a soma de uma série de valores.
- Contador é uma variável usada para contar — nesse caso, contar o número de notas informadas.

3.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador) (Cont.)

- As variáveis utilizadas para armazenar totais normalmente devem ser inicializadas com zero antes de serem usadas em um programa; caso contrário, a soma incluiria o valor anterior armazenado na posição de memória do total.
- As variáveis contadoras normalmente são inicializadas com zero ou um, a depender de seu uso (apresentaremos exemplos que demonstram cada um desses usos).
- Uma variável não inicializada contém um valor de 'lixo' — o último valor armazenado na posição de memória reservada para essa variável..

Erro comum de programação 3.5
Se um contador ou total não for inicializado, os resultados de seu programa provavelmente serão incorretos. Este é um exemplo de um erro lógico.

Dica de prevenção de erro 3.2
Inicialize todos os contadores e totais.

3.8 Formulando algoritmos: estudo de caso 1 (repetição controlada por contador) (Cont.)

- O cálculo da média no programa produziu o número inteiro 81.
- Na realidade, a soma das notas nesse exemplo é 817, que, ao ser dividido por 10, deveria gerar 81,7, ou seja, um número com uma parte fracionária.
- Veremos como lidar com esses números (chamados números em ponto flutuante) na próxima seção.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela)

- Vamos generalizar o problema da média da turma.
- Considere o seguinte problema:
  - *Desenvolva um programa que calcule a média da turma e que processe um número arbitrário de notas cada vez que o programa for executado.*
- No primeiro exemplo de média da turma, o número de notas (10) era conhecido com antecedência.
- Nesse exemplo, não há nenhuma indicação de quantas notas serão digitadas. O programa deve processar um número qualquer de notas.
- Como o programa pode
- determinar quando parar a leitura de notas? Como ele saberá quando calcular e imprimir a média da turma?

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Um modo de resolver esse problema é usar um valor especial, chamado de valor de sentinela (também chamado de valor sinalizador, valor artificial ou valor de flag), para indicar o 'fim de inserção de dados'.
- O usuário, então, digita as notas até que todas as que são válidas sejam inseridas.
- A seguir, digita o valor de sentinela para indicar que a última nota foi inserida.
- A repetição controlada por sentinela é frequentemente chamada de repetição indefinida, porque o número de repetições não é conhecido antes de o loop começar a ser executado.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Naturalmente, o valor de sentinela deve ser escolhido de forma a não ser confundido com um valor aceitável fornecido como entrada.
- Como as notas de um teste normalmente são inteiros não negativos, −1 é um valor de sentinela aceitável para esse problema.
- Desse modo, uma execução do programa para calcular a média da turma poderia processar uma sequência de dados de entrada tal como 95, 96, 75, 74, 89 e −1.
- O programa, então, calcularia e imprimiria a média da turma para as notas 95, 96, 75, 74 e 89 (−1 é o valor de sentinela, e por isso, ele não deve entrar no cálculo da média).

Erro comum de programação 3.6
Escolher um valor de sentinela que também seja um valor de dado legítimo.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Abordaremos o programa que calcula a média da turma com uma técnica chamada refinamentos sucessivos top-down, uma técnica essencial para o desenvolvimento de programas bem-estruturados.
- Começaremos com uma representação do pseudocódigo de cima para baixo:
  - Determinar a média da turma para o teste
- O topo é uma única afirmação que expõe a função global do programa.
- Como tal, o topo é, na realidade, uma representação completa de um programa.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Infelizmente, o topo raramente traz uma quantidade suficiente de detalhes para que se escreva um programa em C.
- Assim, começaremos agora o processo de refinamento.
- Temos de dividir o topo em uma série de tarefas pequenas, e depois listar essas tarefas na ordem em que precisam ser executadas.
- Isso resulta no seguinte primeiro refinamento:
  - Inicializar variáveis
    Ler, somar e contar as notas do teste
    Calcular e imprimir a média da turma
- Aqui, foi usada somente a estrutura de sequência — as etapas listadas devem ser executadas na ordem, uma depois da outra.

Observação sobre engenharia de software 3.3

Cada refinamento, bem como o próprio topo, é uma especificação completa do algoritmo; só o nível de detalhe varia.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Para prosseguir, chegando ao próximo nível de refinamento, o segundo refinamento, adicionamos variáveis específicas.
- Precisamos de um total acumulado dos números, uma contagem de quantos números foram processados, uma variável para receber o valor de cada nota enquanto ela é inserida e uma variável para conter a média calculada.
- O comando de pseudocódigostatement
  - Inicializar variáveis
- pode ser refinado da seguinte forma:
  - Inicializar total como zero
    Inicializar contador como zero

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Observe que somente o total e o contador precisam ser inicializados; as variáveis média e nota (média calculada e entrada do usuário, respectivamente) não precisam ser inicializadas, pois seus valores serão modificados pelo processo de leitura destrutiva, discutido no Capítulo 2.
- A instrução de pseudocódigo
  - Ler, somar e contar as notas do teste
- requer uma estrutura de repetição (ou seja, um loop) que leia cada nota sucessivamente.
- Como não sabemos com antecedência quantas notas deverão ser processadas, usaremos a repetição controlada por sentinela.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- O usuário digitará notas válidas, uma de cada vez.
- Depois que a última nota válida for informada, o usuário digitará o valor de sentinela.
- O programa testará esse valor depois que cada nota for lida e terminará o loop quando a sentinela for digitada.
- O refinamento da instrução de pseudocódigo apresentada será, então,
  - Inserir a primeira nota

    Enquanto o usuário ainda não tiver digitado a sentinela
    Somar essa nota ao total acumulado
    Somar um ao contador de notas
    Ler a próxima nota (possivelmente a sentinela)

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Note que, em pseudocódigo, não usamos chaves em torno do conjunto de comandos que forma o corpo da estrutura *while*.
- Simplesmente recuamos todos esses comandos sob *while* para mostrar que eles pertencem à estrutura.
- Novamente, o pseudocódigo é somente uma ajuda informal para o desenvolvimento de programas.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- O comando de pseudcódico
  - Calcular e imprimir a média da turma
- pode ser refinado da seguinte forma:
  - Se o contador não é igual a zero
    Definir a média como total dividido pelo contador
    Imprimir a média
    se não
    Imprimir 'Nenhuma nota foi informada'
- Note que estamos sendo cuidadosos aqui, testando a possibilidade de divisão por zero — um erro fatal de lógica que, se não for detectado, fará com que o programa seja encerrado prematuramente (diz-se, frequentemente, que programa 'abortaria' ou 'falharia').
- O segundo refinamento completo é mostrado na Figura 3.7.

Erro comum de programação 3.7
Uma tentativa de dividir por zero causa um erro fatal.

Boa prática de programação 3.5
Ao executar uma divisão por uma expressão cujo valor poderia ser zero, teste explicitamente essa possibilidade e trate-a de forma apropriada em seu programa (por exemplo, imprimindo uma mensagem de erro) em vez de permitir que o erro fatal aconteça.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Nas figuras 3.5 e 3.7, incluímos algumas linhas em branco no pseudocódigo para aumentar a legibilidade.
- As linhas em branco, na verdade, separam esses programas em suas várias fases.

```
1  Inicializar total como zero
2  Inicializar contador como zero
3
4  Ler a primeira nota
5  Enquanto o usuário não tiver digitado a sentinela
6          Somar essa nota ao total acumulado
7          Somar um ao contador de notas
8          Ler a próxima nota (possivelmente a sentinela)
9
10 Se o contador não é igual a zero
11         Definir a média como total dividido pelo contador
12         Imprimir a média
13 se não
14         Imprimir "Nenhuma nota foi informada"
```

Figura 3.7 ■ Algoritmo de pseudocódigo que usa a repetição controlada por sentinela para resolver o problema de média da turma.

Observação sobre engenharia de software 3.4
Muitos programas podem ser logicamente divididos em três fases: uma fase de inicialização, que inicializa as variáveis do programa; uma fase de processamento, que recebe como entrada valores de dados e ajusta as variáveis do programa adequadamente; e uma fase de finalização, que calcula e imprime os resultados finais.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- O algoritmo em pseudocódigo na Figura 3.7 resolve a classe mais geral de problemas de cálculo da média.
- Esse algoritmo foi desenvolvido após somente dois níveis de refinamento.
- Às vezes, mais níveis são necessários.

Observação sobre engenharia de software 3.5

Você termina o processo de refinamento sucessivo top-down quando o algoritmo em pseudocódigo estiver especificado em detalhes suficientes para que você possa converter o pseudocódigo em C. Implementar o programa em C normalmente será um processo direto.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- O programa em C e um exemplo de sua execução são mostrados na Figura 3.8.
- Embora somente notas com números inteiros sejam fornecidas, é provável que o cálculo da média produza um número decimal (com uma parte inteira e outra fracionária).
- O tipo `int` não pode representar números reais.
- O programa introduz o tipo de dados `float` para tratar números com ponto decimal (também chamados de números em ponto flutuante) e introduz um operador especial chamado de operador de conversão (cast) para tratar do cálculo da média.
- Esses recursos são explicados em detalhes depois de o programa ser apresentado

```
1  /* Figura 3.8: fig03_08.c
2     Programa de média da turma com repetição controlada por sentinela */
3  #include <stdio.h>
4
5  /* função main inicia execução do programa */
6  int main( void )
7  {
```

Figura 3.8 ■ Programa em C e exemplo de sua execução para o problema da média da turma com repetição controlada por sentinela. (Parte 1 de 2.)

```
8     int contador;  /* número de notas digitadas */
9     int nota;  /* valor da nota */
10    int total;  /* soma das notas */
11
12    float média;  /* número em ponto flutuante para a média */
13
14    /* fase de inicialização */
15    total = 0;  /* inicializa total */
16    contador = 0;  /* inicializa contador do loop */
17
18    /* fase de processamento */
19    /* recebe primeira nota do usuário */
20    printf( "Digite a nota, -1 no fim: " ); /* prompt para entrada */
21    scanf( "%d", &nota ); /* lê nota do usuário */
22
23    /* loop enquanto valor da sentinela não foi lido */
24    while ( nota != -1 ) {
25       total = total + nota;     /* soma nota ao total */
26       contador = contador + 1;  /* incrementa contador */
27
28       /* recebe próxima nota do usuário */
29       printf( "Digite a nota, -1 para finalizar: " ); /* prompt para entrada */
30       scanf("%d", &nota); /* lê próxima nota */
31    } /* fim do while */
32
33    /* fase de finalização */
34    /* se o usuário digitou pelo menos uma nota */
35    if ( contador != 0 ) {
36
```

```
37          /* calcula média de todas as notas lidas */
38          média = ( float ) total / contador; /* evita truncar */
39
40          /* exibir média com dois dígitos de precisão */
41          printf( "Média da turma é %.2f\n", média );
42       } /* fim do if */
43       else { /* se nenhuma nota foi informada, envia mensagem */
44          printf( "Nenhuma nota foi informada\n" );
45       } /* fim do else */
46
47       return 0; /* indica que o programa foi concluído com sucesso */
48    } /* fim da função main */
```

```
Digite a nota, -1 para finalizar: 75
Digite a nota, -1 para finalizar: 94
Digite a nota, -1 para finalizar: 97
Digite a nota, -1 para finalizar: 88
Digite a nota, -1 para finalizar: 70
Digite a nota, -1 para finalizar: 64
Digite a nota, -1 para finalizar: 83
Digite a nota, -1 para finalizar: 89
Digite a nota, -1 para finalizar: -1
Média da turma é 82,50
```

```
Digite a nota, -1 no fim: -1
Nenhuma nota foi informada
```

Figura 3.8 ■ Programa em C e exemplo de sua execução para o problema da média da turma com repetição controlada por sentinela. (Parte 2 de 2.)

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Note o comando composto no loop `while` na Figura 3.8. Mais uma vez, as chaves são necessárias para as quatro instruções a serem executadas dentro do loop.
- Sem as chaves, os últimos três comandos no corpo do loop cairiam para fora dele, fazendo com que o computador interpretasse incorretamente esse código, como a seguir:

```
while ( nota != -1 )
   total = total + nota; /* soma nota ao total */
contador = contador + 1; /* incrementa contador */
printf( "Digite a nota, -1 no fim: " ); /* prompt da entrada */
scanf( "%d", &nota ); /* lê próxima nota */
```

- Isso causaria um loop infinito se o usuário não fornecesse −1 como entrada para a primeira nota.

Boa prática de programação 3.6

Em um loop controlado por sentinela, os prompts que solicitam a entrada de dados deveriam lembrar explicitamente ao usuário qual o valor de sentinela.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Cálculos de médias nem sempre produzem valores inteiros.
- Frequentemente, uma média é um valor que contém uma parte fracionária, como 7,2 ou −93,5.
- Esses valores são chamados de números em ponto flutuante, e são representados pelo tipo de dados `float`.
- A variável `média` é declarada para ser do tipo `float` (linha 12) para capturar o resultado de nosso cálculo em ponto flutuante.
- Porém, o resultado do cálculo `total / contador` é um número inteiro, porque `total` e `contador` asão ambas váriaveis inteiras.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Dividir dois inteiros resulta em uma divisão inteira em que qualquer parte fracionária do cálculo é perdida (isto é, truncada).
- Como o cálculo é executado primeiro, a parte fracionária é perdida antes de o resultado ser atribuído à `média`.
- Para produzir um cálculo de ponto flutuante com valores inteiros, devemos criar valores temporários que sejam números em ponto flutuante.
- C fornece o operador unário de conversão para realizar essa tarefa.

- A linha 38
  - `média = ( float ) total / contador;`
- inclui o operando de conversão `(float)`, que cria uma cópia de ponto flutuante temporária de seu operando, `total`.
- O valor armazenado em `total` ainda é um inteiro.
- O uso de um operador de conversão dessa maneira é chamado de conversão explícita.
- O cálculo agora consiste em um valor de ponto flutuante (a versão `float` temporária de `total`) dividido pelo inteiro `contador`.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- A maioria dos computadores só pode avaliar expressões aritméticas em que os tipos de dados dos operandos são idênticos.
- Para assegurar que os operandos sejam do mesmo tipo, o compilador executa uma operação chamada promoção (também denominada conversão implícita) sobre operandos selecionados.
- Por exemplo, em uma expressão contendo os tipos de dados `int` e `float`, cópias de operandos `int` são promovidas a `float`.
- O cálculo é executado e o resultado da divisão em ponto flutuante é atribuído à `média`.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Capítulo 5 apresenta uma discussão de todos os tipos de dados-padrão e sua ordem de promoção.
- Os operadores de conversão estão disponíveis para qualquer tipo de dados.
- O operador de conversão é formado colocando-se parênteses em torno do nome de um tipo de dados.
- O operador de conversão é um operador unário, ou seja, um operador que aceita somente um operando.
- No Capítulo 2, estudamos os operadores aritméticos binários.
- C também suporta versões unárias dos operadores mais (+) e menos (−), de modo que você possa escrever expressões como −7 ou +5.

- Operadores de conversão são avaliados da direita
- para a esquerda, e têm a mesma precedência de outros operadores unários, como o + unário e o − unário.
- Essa precedência é mais alta que a dos operadores multiplicativos *, / e %.
- A Figura 3.8 usa o especificador de conversão de `printf %.2f` (linha 41) para imprimir o valor de `média`.
- O `f` especifica que um valor de ponto flutuante será impresso.
- O `.2` é a precisão com que o valor será exibido — com 2 dígitos à direita do ponto decimal.

- Se o especificador de conversão `%f` for usado (sem especificar a precisão), a precisão default de 6 é usada — exatamente como se o especificador de conversão `%.6f` tivesse sido usado.
- Quando os valores de ponto flutuante são impressos com precisão, o valor impresso é arredondado para o número de posições decimais indicado.
- O valor na memória não é alterado.
- Quando as seguintes instruções sãoex ecutadas, os valores 3,45 e 3,4 são impressos..

```
printf( "%.2f\n", 3.446 ); /* imprime 3.45 */
printf( "%.1f\n", 3.446 ); /* imprime 3.4  */
```

Erro comum de programação 3.8
Usar precisão em uma especificação de conversão na string de controle de formato de uma instrução scanf é errado. As precisões são usadas apenas nas especificações de conversão de printf.

Erro comum de programação 3.9
Usar números em ponto flutuante de uma maneira que sugira que eles sejam representados com precisão pode ocasionar resultados incorretos. Os números em ponto flutuante são representados apenas aproximadamente pela maioria dos computadores.

Dica de prevenção de erro 3.3
Não compare valores em ponto flutuante quanto à igualdade ou desigualdade.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Apesar do fato de os números em ponto flutuante não serem sempre ‘100 por cento precisos’, eles têm inúmeras aplicações.
- Por exemplo, quando dizemos que a temperatura ‘normal’ do corpo é 36,7 °C, não temos de ser precisos em um grande número de dígitos.
- Quando olhamos a temperatura em um termômetro e lemos 36,7, ela realmente poderia ser 36.6999473210643.
- A questão a ser destacada aqui é que chamar esse número simplesmente de 36,7 é suficiente para a maioria das aplicações.

3.9 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 2 (repetição controlada por sentinela) (Cont.)

- Números em ponto flutuante também surgem por meio da divisão.
- Quando dividimos 10 por 3, o resultado é 3,3333333... com a sequência de 3 repetindo-se infinitamente.
- O computador aloca uma quantidade fixa de espaço para guardar esse valor; assim, obviamente, o valor armazenado em ponto flutuante só pode ser uma aproximação.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas)

- Estudaremos agora outro problema completo.
- Uma vez mais, formularemos o algoritmo usando pseudocódigo e refinamentos sucessivos top-down, e escreveremos um programa correspondente em C.
- Vimos que estruturas de controle podem ser empilhadas umas sobre as outras (em sequência) da mesma maneira que uma criança empilha blocos de montagem.
- Nesse estudo de caso, veremos outro modo estruturado pelo qual estruturas de controle podem ser conectadas em C: o aninhamento de uma estrutura de controle dentro de outra..

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Considere a seguinte definição de problema:
  - Uma escola oferece um curso que prepara alunos para o exame estadual de licenciamento de corretores de imóveis. No último ano, dez dos alunos que completaram esse curso fizeram o exame de licenciamento. Naturalmente, a escola quer saber como os alunos se saíram no exame. Você recebeu a missão de escrever um programa que resuma os resultados. Você recebeu uma lista com os nomes dos dez alunos. Ao lado de cada nome, lê-se 1 se o aluno passou no exame, ou 2, se o aluno foi reprovado.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Seu programa deve analisar os resultados do exame da seguinte forma:
  - Forneça como entrada cada resultado do teste (isto é, 1 ou 2). Exiba a mensagem 'Digite o resultado' na tela cada vez que o programa pedir outro resultado do teste.
  - Conte o número de resultados de teste de cada tipo.
  - Exiba um resumo dos resultados de teste que indique o número de alunos que passaram e o número de alunos que foram reprovados.
  - Se mais de oito alunos tiverem passado no exame, imprima a mensagem 'Bônus para o instrutor!'

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Depois de ler a definição do problema cuidadosamente, temos as seguintes observações:
  - O programa deve processar 10 resultados do teste. Um loop controlado por contador deverá ser usado.
  - Cada resultado do teste é um número — 1 ou 2. Cada vez que o programa lê um resultado, ele deve determinar se o número é 1 ou 2. Em nosso algoritmo, testamos se ele é 1. Se o número não for 1, iremos supor que ele seja 2 (um exercício ao fim do capítulo aborda as consequências dessa suposição).
  - Dois contadores são usados: um para contar o número de alunos que passaram no exame, e outro para contar o número de alunos que foram reprovados.
  - Depois de o programa ter processado todos os resultados, ele deve decidir se mais de 8 alunos passaram no exame.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Continuaremos com o refinamento sucessivo top-down e com uma representação em pseudocódigo de auto nível:
  - Analisar os resultados do exame e decidir se o instrutor deve receber um bônus
- Uma vez mais, é importante enfatizar que o topo é uma representação completa do programa, mas, provavelmente, vários refinamentos serão necessários antes que o pseudocódigo possa ser naturalmente convertido em um programa em C..

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Nosso primeiro refinamento é
  - Inicializar variáveis
    Obter as dez notas do teste e contabilizar aprovações e reprovações
    Imprimir um resumo dos resultados do teste e decidir se o instrutor deve receber um bônus
- Um refinamento adicional é necessário nesse ponto, embora tenhamos uma representação completa de todo o programa.
- Agora definiremos variáveis específicas.
- Serão necessários contadores para registrar aprovações e reprovações, um contador será usado para controlar o loop de processamento, e é preciso que haja uma variável que armazene a entrada fornecida pelo usuário.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- O comando de pseudocódigo
  - Inicializar variáveis
- pode ser refinado da seguinte maneira:
  - Inicializar aprovações como zero
    Inicializar reprovações como zero
    Inicializar aluno como um
- Note que somente os contadores e totais são inicializados.
- O comando de pseudocódigo
  - Obter as dez notas do teste e contabilizar aprovações e reprovações
- exige um loop, que sucessivamente recebe como entrada o resultado de cada exame.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Nesse momento, sabemos com antecedência que existem exatamente dez resultados do teste; assim, um loop controlado por contador é adequado.
- Dentro do loop (ou seja, aninhada dentro do loop), uma estrutura de seleção dupla determinará se cada resultado do teste é uma aprovação ou uma reprovação e, consequentemente, incrementará o contador apropriado de modo adequado.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- O refinamento do comando de pseudocódigo apresentado, então, toma a seguinte forma:
  - Enquanto o contador de alunos é menor ou igual a dez
    Lê o próximo resultado do exame

    Se o aluno passou
    Soma um às aprovações
    se não
    Soma um às reprovações

    Soma um ao contador de alunos
- Note o uso de linhas em branco para separar a estrutura de controle se...se não para melhorar a legibilidade do programa.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- O comando de pseudocódigo
  - Imprimir um resumo dos resultados do teste e decidir se o instrutor deve receber um bônus
- pode ser refinado da seguinte forma:
  - Imprimir o número de aprovações
    Imprimir o número de reprovações
    Se mais de oito alunos passaram
    Imprimir "Bônus ao instrutor!"
- O segundo refinamento completo aparece na Figura 3.9.
- Observe que linhas em branco também são usadas aqui para separar a estrutura `while` e melhorar a legibilidade do programa.

3.10 Formulando algoritmos com refinamentos sucessivos top-down: estudo de caso 3 (estruturas de controle aninhadas) (Cont.)

- Agora, esse pseudocódigo está suficientemente refinado para a conversão em C.
- O programa em C e dois exemplos de execução aparecem na Figura 3.10.
- Tiramos proveito de um recurso da linguagem C que permite que a inicialização seja incorporada nas declarações.
- Essa inicialização ocorre durante a compilação.

```
1  Inicializa aprovações como zero
2  Inicializa reprovações como zero
3  Inicializa aluno como um
4
5  Enquanto contador de alunos for menor ou igual a dez
6          Lê o próximo resultado do exame
7
8          Se o aluno passou
9              Soma um às aprovações
10         se não
11             Soma um às reprovações
12
13         Soma um ao contador de alunos
14
15 Imprimir o número de aprovações
16 Imprimir o número de reprovações
17 Se mais de oito alunos passaram
18         Imprimir 'Bônus ao instrutor!'
```

Figura 3.9 ■ Pseudocódigo para o problema de resultados do exame.

Dica de desempenho 3.1

Inicializar variáveis quando elas são declaradas pode ajudar a reduzir o tempo de execução de um programa.

Dica de desempenho 3.2

Muitas das dicas de desempenho que mencionamos no texto resultam em melhorias nominais, de modo que o leitor pode se sentir tentado a ignorá-las. O efeito acumulado de todas essas melhorias de desempenho pode fazer um programa ser executado muito mais rapidamente. Além disso, uma melhoria significativa é observada quando uma melhoria supostamente nominal é colocada em um loop que pode se repetir um grande número de vezes.

```c
/* Figura 3.10: fig03_10.c
   Análise de resultados de exame */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   /* inicializa variáveis nas declarações */
   int aprovados = 0; /* número de aprovações */
   int reprovados = 0; /* número de reprovações */
   int aluno = 1; /* contador de alunos */
   int resultado; /* um resultado de exame */

   /* processa 10 alunos usando loop controlado por contador */
   while ( aluno <= 10 ) {

      /* pede entrada do usuário e lê o valor digitado */
      printf( "Forneça resultado ( 1=aprovado,2=reprovado): " );
      scanf( "%d", &resultado );

      /* se resultado 1, incrementa aprovados */
      if ( resultado == 1 ) {
      aprovados = aprovados + 1;
      } /* fim do if */
      else {/* senão, incrementa reprovados */
      reprovados = reprovados + 1;
      } /* fim do else */

```

```
29          aluno = aluno + 1; /* incrementa contador de alunos */
30       } /* fim do while */
31
32       /* fase de finalização; exibe número de aprovados e reprovados */
33       printf( "Aprovados %d\n", aprovados );
34       printf( "Reprovados %d\n", reprovados );
35
36       /* se mais de oito aprovados, imprime 'Bônus ao instrutor!' */
37       if ( aprovados > 8 ) {
38       printf( "Bônus ao instrutor!\n" );
39       } /* fim do if */
40
41       return 0; /* indica que o programa foi concluído com sucesso */
42    } /* fim da função main */
```

```
Forneça resultado (1=aprovado, 2=reprovado):  1
Forneça resultado (1=aprovado, 2=reprovado):  2
Forneça resultado (1=aprovado, 2=reprovado):  2
Forneça resultado (1=aprovado, 2=reprovado):  1
Forneça resultado (1=aprovado, 2=reprovado):  1
Forneça resultado (1=aprovado, 2=reprovado):  1
Forneça resultado (1=aprovado, 2=reprovado):  2
Forneça resultado (1=aprovado, 2=reprovado):  1
Forneça resultado (1=aprovado, 2=reprovado):  1
Forneça resultado (1=aprovado, 2=reprovado):  2
Aprovados 6
Reprovados 4
```

Figura 3.10 ■ Programa em C e exemplos de sua execução para o problema dos resultados de exame. (Parte 1 de 2.)

Observação sobre engenharia de software 3.6
A experiência tem mostrado que a parte mais difícil da solução de um problema em um computador é desenvolver o algoritmo dessa solução. Uma vez que o algoritmo correto tiver sido especificado, o processo de produção de um programa funcional em C normalmente será simples.

Observação sobre engenharia de software 3.7
Muitos programadores escrevem programas sem sequer usar ferramentas de desenvolvimento, tais como o pseudocódigo. Eles acham que seu objetivo final é resolver o problema em um computador, e que escrever pseudocódigos simplesmente atrasa a obtenção do resultado final.

3.11 Operadores de atribuição

- C oferece vários operadores de atribuição que abreviam as expressões de atribuição.
- Por exemplo, a instrução
  - `c = c + 3;`
- pode ser abreviada com o operador de atribuição com adição += para
  - `c += 3;`
- O operador += soma o valor da expressão à direita do operador ao valor da variável à esquerda do operador, e armazena o resultado na variável à esquerda do operador.

3.11 Operadores de atribuição (Cont.)

- Qualquer instrução da forma
  - *variável = variável operador expressão;*
- onde *operador* é um dos operadores binários +, −, *, / ou % (ou outros que serão discutidos no Capítulo 10), pode ser escrita na forma
  - *variável operador = expressão* **;**
- Assim, a atribuição c += 3 soma 3 a c.
- A Figura 3.11 mostra os operadores aritméticos de atribuição e exemplos de expressões que usam esses operadores e explicações.

Operador de atribuição	Exemplo de expressão	Explicação	Atribui
Considere: **int** c = 3, d = 5, e = 4, f = 6, g = 12;			
+=	c += 7	c = c + 7	10 a c
-=	d -= 4	d = d - 4	1 a d
*=	e *= 5	e = e * 5	20 a e
/=	f /= 3	f = f / 3	2 a f
%=	g %= 9	g = g % 9	3 a g

Figura 3.11 ■ Operadores aritméticos de atribuição.

3.12 Operadores de incremento e decremento

- C também fornece o operador unário de incremento ++ e o operador unário de decremento −−, que estão resumidos na Figura 3.12.
- Se uma variável c é incrementada em 1, o operador de incremento + + pode ser usado no lugar das expressões c = c + 1 ou c + = 1.
- Se um operador de incremento ou de decremento for colocado antes de uma variável (ou seja, se ele for prefixado), ele é chamado de operador de pré-incremento ou de pré-decremento, respectivamente.
- Se um operador de incremento ou decremento é colocado depois de uma variável, ele é chamado de operador de pós-incremento ou de pós-decremento, respectivamente.

3.12 Operadores de incremento e decremento (Cont.)

- Pré-incrementar (ou pré-decrementar) uma variável faz com que a variável seja incrementada (ou decrementada) em 1, sendo o novo valor da variável, então, usado na expressão em que ela aparece.
- Pós-incrementar (ou pós-decrementar) uma variável faz com que o valor corrente da variável seja primeiramente usado na expressão em que ela aparece, para depois ser incrementado (ou decrementado) por 1.

Operador	Exemplo de expressão	Explicação
++	++a	Incrementa a em 1, e então usa o novo valor de a na expressão em que a estiver.
++	a++	Usa o valor corrente de a na expressão em que a estiver, e então incrementa a em 1.
--	--b	Decrementa b em 1, e então usa o novo valor de b na expressão em que b estiver.
--	b--	Usa o valor corrente de b na expressão em que b estiver, e então decrementa b em 1.

Figura 3.12 ■ Operadores de incremento e decremento.

3.12 Operadores de incremento e decremento (Cont.)

- O programa da Figura 3.13 mostra a diferença entre a versão de pré-incremento e a versão de pós-incremento do operador ++.
- Pós-incrementar a variável `c` faz com que ela seja incrementada depois de ser usada no comando de `printf`.
- Pré-incrementar a variável `c` faz com que ela seja incrementada antes de ser usada no comando `printf`.

```c
/* Figura 3.13: fig03_13.c
   Pré-incrementando e pós-incrementando */
#include <stdio.h>

/* função main inicia a execução do programa */
int main( void )
{
   int c; /* define variável */

   /* demonstra pós-incremento */
   c = 5; /* atribui 5 a c */
   printf( "%d\n", c ); /* imprime 5 */
   printf( "%d\n", c++ ); /* imprime 5 e depois pós-incrementa*/
   printf( "%d\n\n", c ); /* imprime 6 */

   /* demonstra pré-incremento */
   c = 5; /* atribui 5 a c */
   printf( "%d\n", c ); /* imprime 5 */
```

Figura 3.13 ■ Diferenciando pré-incrementação de pós-incrementação. (Parte 1 de 2.)

```
19      printf( "%d\n", ++c ); /* pré-incrementa e depois imprime 6 */
20      printf( "%d\n", c ); /* imprime 6 */
21      return 0; /* indica que o programa foi concluído com sucesso */
22  } /* fim da função main */
```

```
5
5
6
 s
5
6
6
```

Figura 3.13 ■ Diferenciando pré-incrementação de pós-incrementação. (Parte 2 de 2.)

3.12 Operadores de incremento e decremento (Cont.)

- O programa exibe o valor de c antes e depois que o operador ++ é usado.
- O operador de decremento (−−) funciona de modo semelhante.

Boa prática de programação 3.7
Operadores unários devem ser colocados perto de seus operandos, e espaços intermediários não devem ser usados.

3.12 Operadores de incremento e decremento (Cont.)

- Os três comandos de atribuição na Figura 3.10
  - ```
    aprovados = aprovados + 1;
    reprovados = reprovados + 1;
    aluno = aluno + 1;
    ```

  poderão ser escritos de forma mais concisa se usarmos operadores de atribuição, tais como
  - ```
    aprovados += 1;
    reprovados += 1;
    aluno += 1;
    ```

  usando operadores de pré-incremento como
  - ```
    ++aprovados;
    ++reprovados;
    ++aluno;
    ```

  ou usando operadores de pós-incremento como
  - ```
    aprovados++;
    reprovados++;
    aluno++;
    ```

3.12 Operadores de incremento e decremento (Cont.)

- Note que, ao incrementar ou decrementar uma variável sozinha em uma instrução, as formas de pré-incremento e pós-incremento têm o mesmo efeito.

3.12 Operadores de incremento e decremento (Cont.)

- Somente quando uma variável aparece no contexto de uma expressão maior é que os efeitos são diferentes (e, similarmente, para pré-decrementar e pós-decrementar).
- Das expressões que estudamos até aqui, somente um nome de variável simples pode ser usado como operando de um operador de incremento ou decremento.

Erro comum de programação 3.10
Tentar usar um operador de incremento ou decremento em uma expressão que não seja um nome de variável simples é um erro de sintaxe; escrever ++(x + 1), por exemplo.

Dica de prevenção de erro 3.4
C geralmente não especifica a ordem em que os operandos de um operador serão avaliados (veremos exceções a essa regra para alguns operadores no Capítulo 4). Portanto, você deve evitar usar instruções com operadores de incremento ou decremento em que uma variável em particular, sendo incrementada ou decrementada, aparece mais de uma vez.

3.12 Operadores de incremento e decremento (Cont.)

- A Figura 3.14 mostra a precedência e a associatividade dos operadores apresentados até aqui.
- Os operadores são mostrados de cima para baixo em ordem decrescente de precedência.
- A segunda coluna descreve a associatividade dos operadores em cada nível de precedência.
- Observe que o operador condicional (?:), os operadores unários incremento (++) e decremento (––), mais (+), menos (–) e de conversão, e os operadores de atribuição =, +=, –=, *=, /= e %= se associam da direita para a esquerda.
- A terceira coluna nomeia os vários grupos de operadores.
- Todos os outros operadores na Figura 3.14 se avaliam da esquerda para a direita.

Operadores	Associatividade	Tipo
++ (*pós-fixo*) -- (*pós-fixo*)	direita para esquerda	pós-fixo
+ - (*tipo*) ++ (*prefixo*) -- (*prefixo*)	direita para esquerda	unário
* / %	esquerda para direita	multiplicativo
+ -	esquerda para direita	aditivo
< <= > >=	esquerda para direita	relacional
== !=	esquerda para direita	igualdade
?:	direita para esquerda	condicional
= += -= *= /= %=	direita para esquerda	atribuição

Figura 3.14 ■ Precedência e associatividade dos operadores encontrados até agora no texto.