Lógica e Design de Programação: Introdução

Capítulo 5

Looping

Objetivos

- Compreendendo as vantagens de usar os loops
- Controlando loops com contadores e flags
- Loops embutidos
- Evitando erros comuns com loops

Objetivos (cont.)

- Usando um loop `for`
- Usando loops pós-teste
- Reconhecendo as características compartilhadas por todos os loops
- Aplicações comuns de loops

Compreendendo as vantagens de usar os loops

- O uso de loops é que faz a programação de computadores valer a pena e ser eficiente
- Usa-se um loop em um programa de computador quando se escreve um conjunto de instruções para operar em múltiplos conjuntos separados de dados
- **Loop**: estrutura que repete ações enquanto determinada condição se mantiver

Compreendendo as vantagens de usar os loops (cont.)

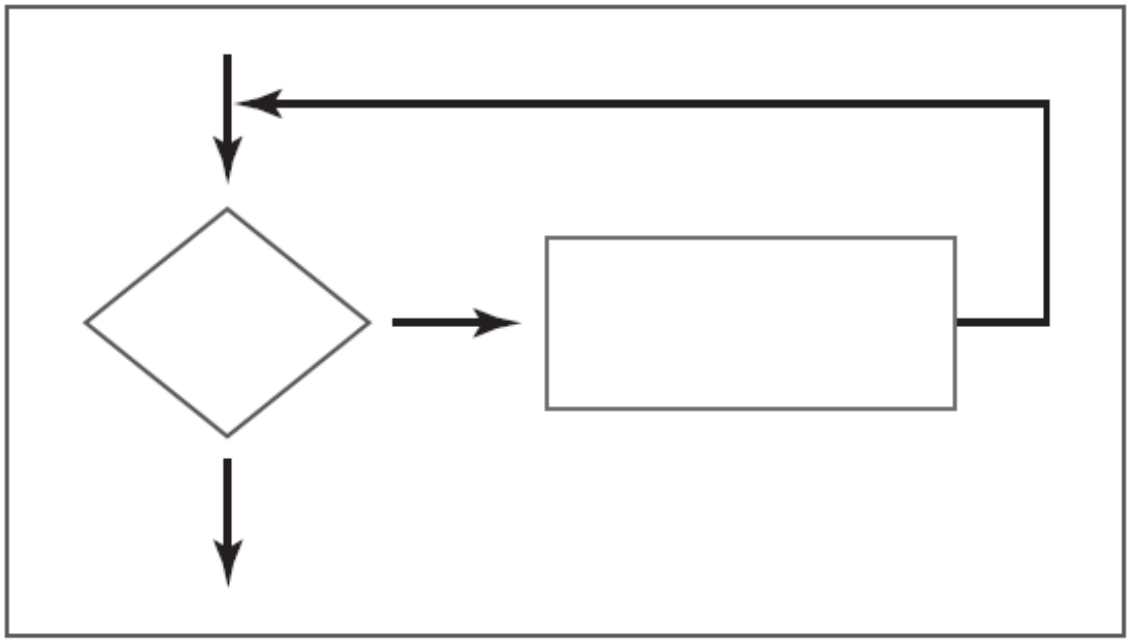


Figura 5-1 O loop `enquanto`

Controlando loops com contadores e flags

- Enquanto uma expressão booleana continuar verdadeira, o corpo de um loop `enquanto` executa
- É preciso controlar o número de repetições que o loop executa
- Normalmente as repetições de um loop são controladas por meio de um contador ou um flag

Usando um loop `enquanto` definido com um contador

- Para fazer um loop enquanto terminar corretamente, três ações separadas devem ocorrer:
  - Uma variável, a **variável de controle do loop**, é inicializada (antes de entrar no loop)
  - A variável de controle do loop é testada e, se o resultado for verdadeiro, entra-se no corpo do loop
  - O corpo do loop deve fazer alguma ação que altere o valor da variável de controle do loop, para que a expressão enquanto, em algum momento, seja avaliada como falsa
  - Muitos valores de variáveis de controle dos loops são alterados por:
    - **Incremento**
    - **Decremento**

Usando um loop enquanto definido com um contador (cont.)

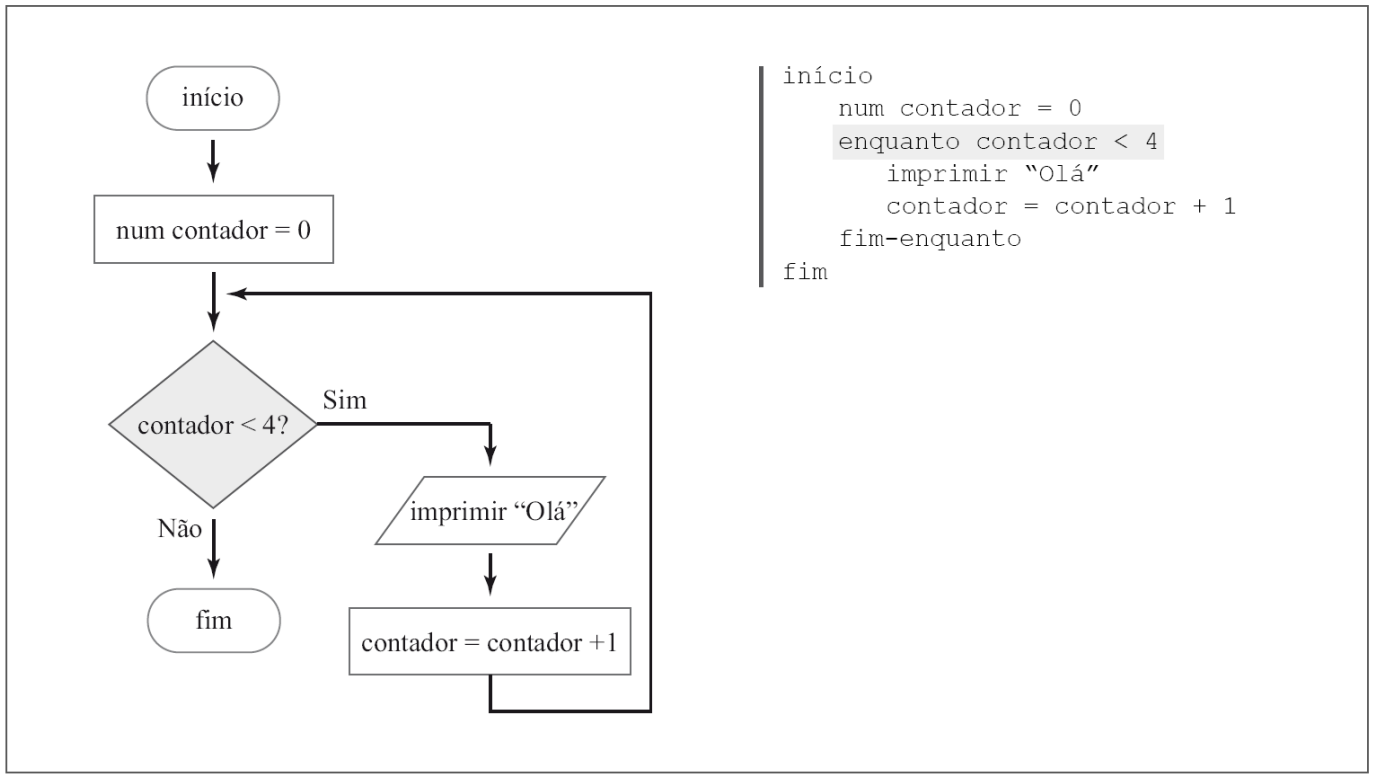


Figura 5-2 Um loop `enquanto` que imprime "Olá" quatro vezes

Usando um loop `enquanto` definido com um contador (cont.)

- **Loop definido** ou **loop contado**: número de iterações é predeterminado
- **Contador**: qualquer variável numérica que conta o número de vezes que um evento ocorreu
- Muitas vezes, o valor de uma variável de controle do loop não é alterado por cálculos, mas por entradas do usuário
- **Loop indefinido**: quando não é possível saber se o loop será executado duas, duzentas ou nenhuma vez

Usando um loop *enquanto* indefinido com um valor flag

- Loop indefinido: cada vez que o programa executa, o loop será realizado um diferente número de vezes
- Três etapas que precisam ocorrer em qualquer loop:
  - É necessário fornecer um valor inicial, que vai controlar o loop
  - É necessário fazer uma comparação usando o valor de controle para verificar se o loop continua ou para
  - Dentro do loop, é necessário alterar o valor que controla o loop

Usando um loop *enquanto* indefinido com um valor flag

- Em cada etapa do loop, o valor da variável resposta determina se o loop continuará
  - Portanto, variáveis como resposta são conhecidas como **variáveis de controle do loop**

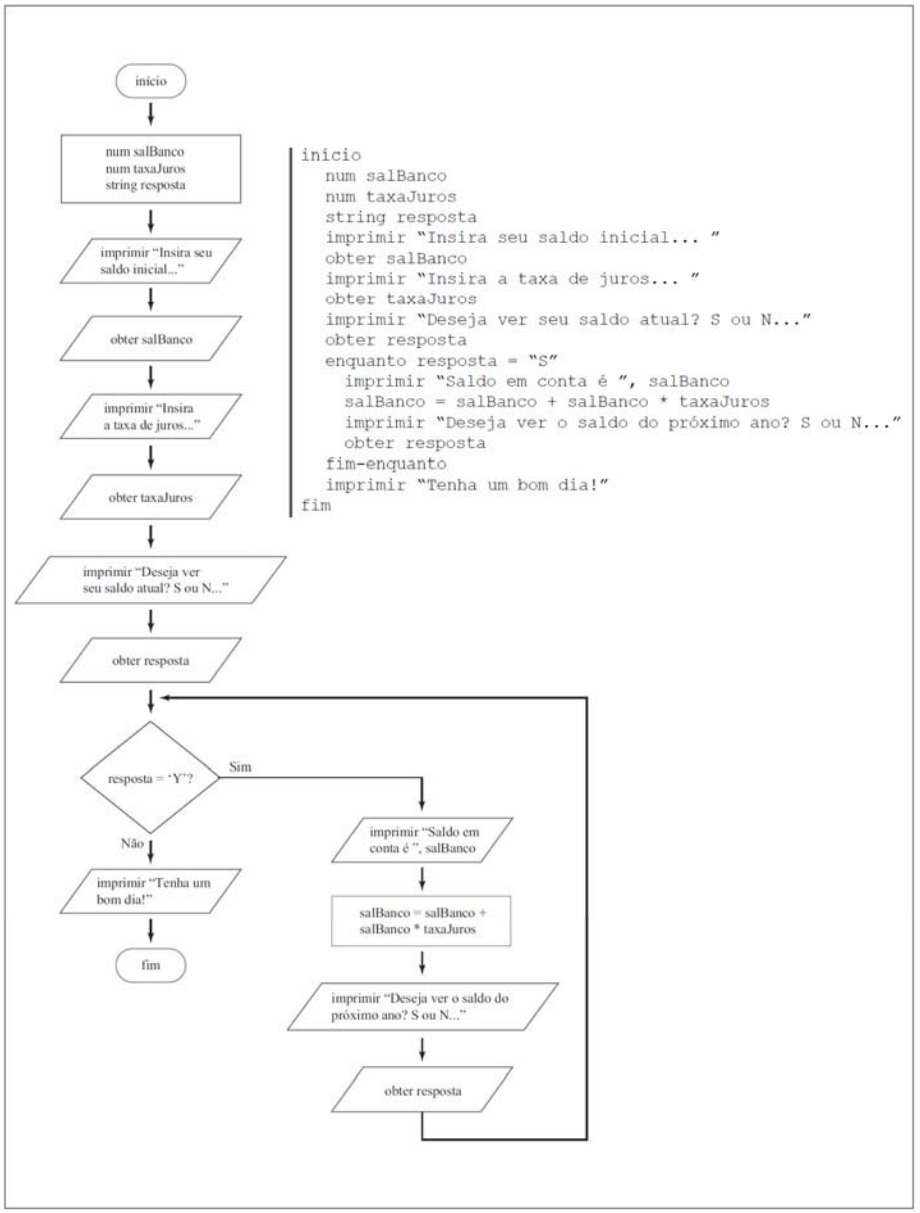


Figura 5-3 Programa de looping de saldo bancário

Usando um loop *enquanto* indefinido com um valor flag (cont.)

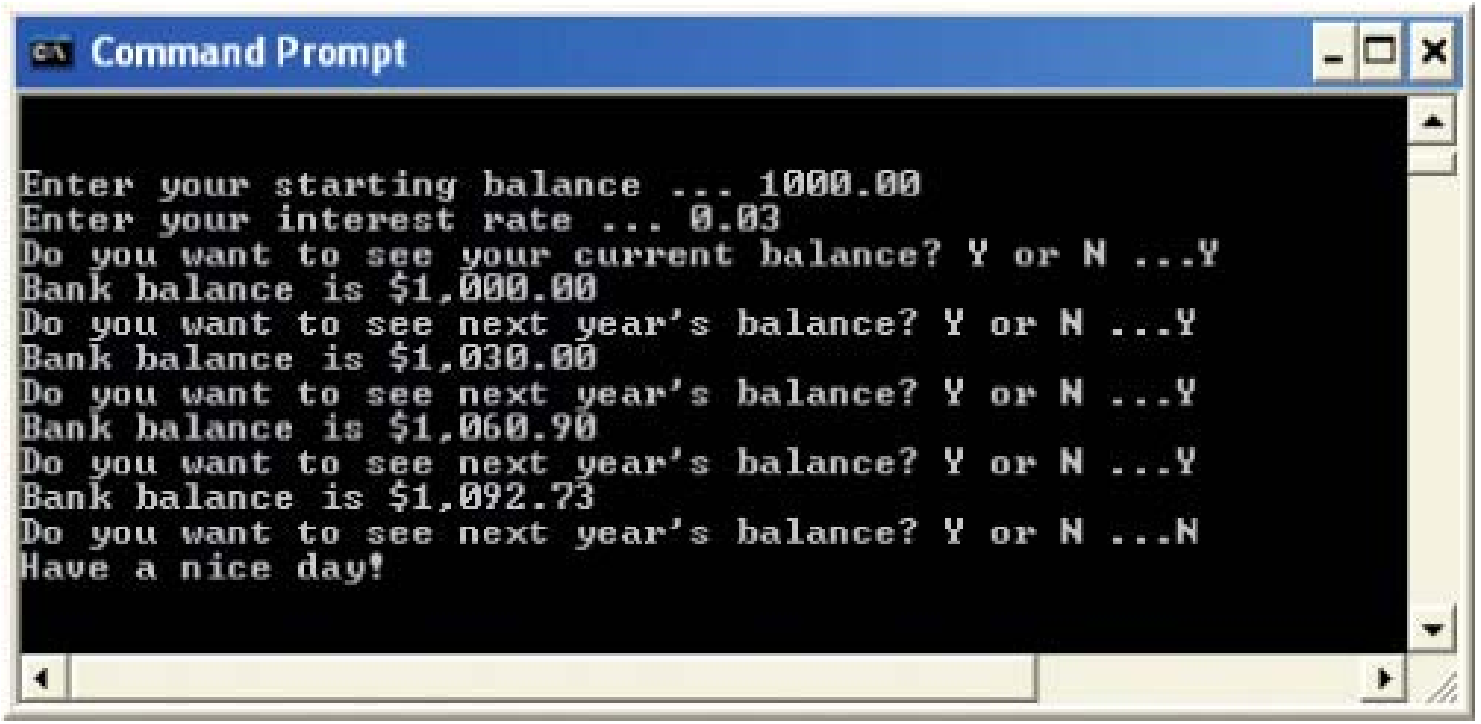


Figura 5-4 Execução típica do programa de looping de saldo bancário

Usando um loop *enquanto* indefinido com um valor flag (cont.)

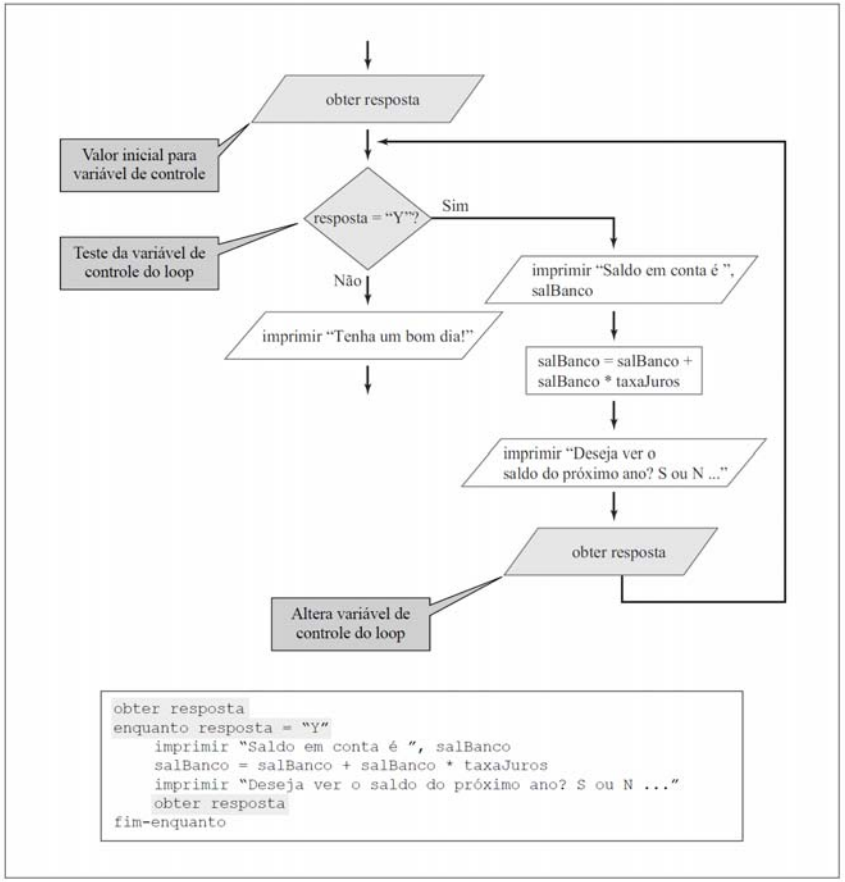


Figura 5-5 Etapas cruciais que precisam ocorrer em todo loop

Loops embutidos

- **Loops embutidos**: loops dentro de loops
- **Loop externo**: o loop que contém o outro
- **Loop interno**: loop que está contido
- Você precisa criar loops embutidos quando os valores de duas (ou mais) variáveis se repetem para produzir valores combinados

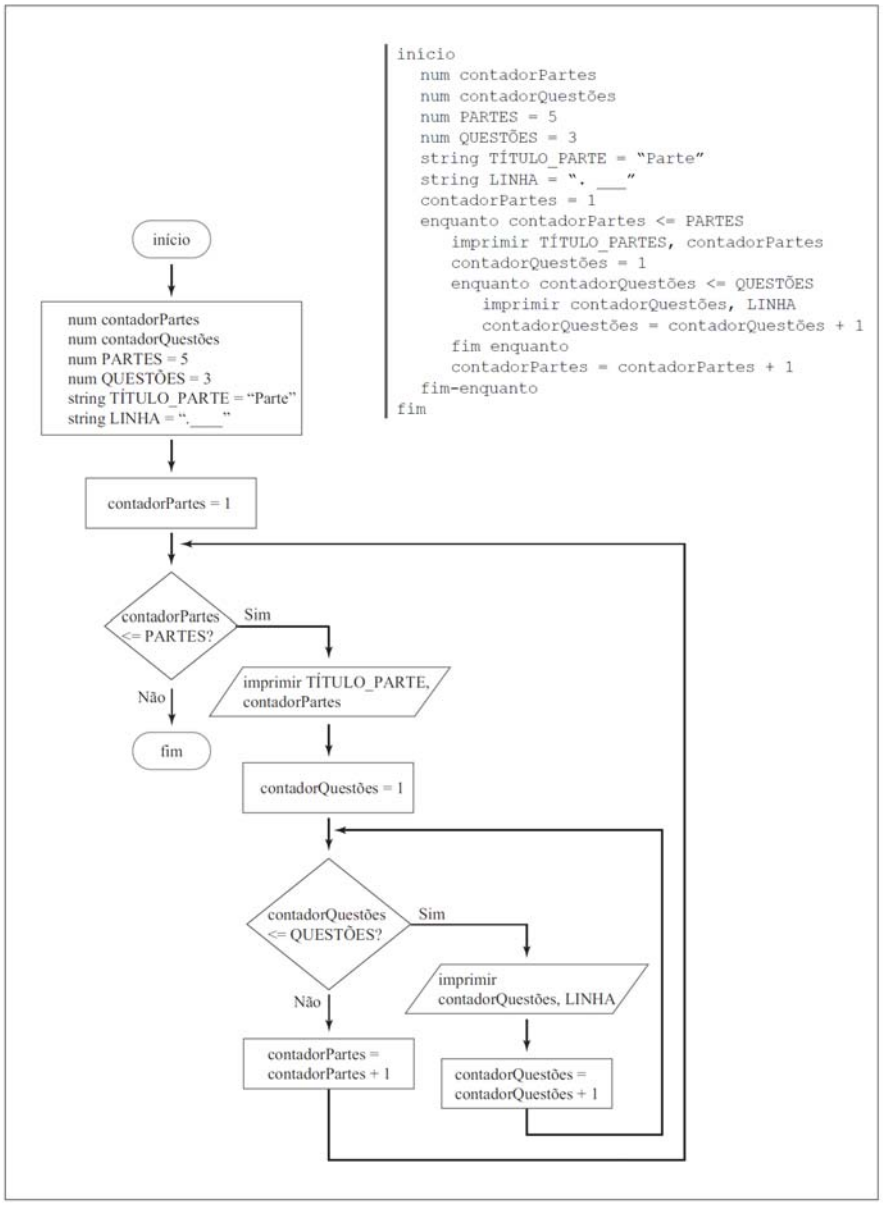


Figura 5-7 Fluxograma e pseudocódigo para programa `folhaRespostas`

Misturando constantes e flags variáveis

- O número de vezes que um loop executa pode depender de uma constante ou de um valor variável
- Às vezes você não quer ser forçado a repetir todos os passos dentro de um loop a mesma quantidade de vezes
- Exemplo:
  - Imprimir 100 rótulos para cada empregado
    - Loop externo é controlado pelo valor do nome do empregado
    - Loop interno executa 100 vezes
  - Número variável de rótulos impressos para cada empregado
    - Loop externo é controlado pelo valor do nome do empregado
    - Loop interno é controlado pelo nível de produção

Misturando constantes e flags variáveis (cont.)

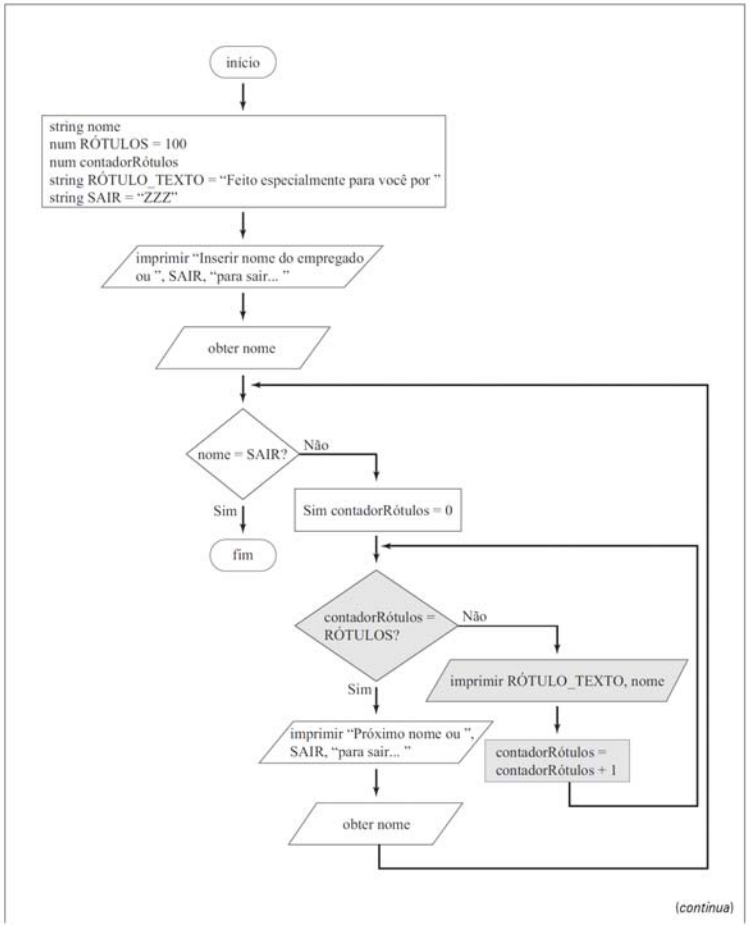


Figura 5-8 Programa que produz cem rótulos com o nome de cada empregado

Misturando constantes e flags variáveis (cont.)

```
início
    string nome
    numérico RÓTULOS = 100
    numérico contadorRótulos
    string RÓTULO_TEXTO = "Feito especialmente para você por "
    string SAIR = "ZZZ"
    imprimir "Inserir nome do empregado ou ", SAIR, "para sair... "
    obter nome
       enquanto nome não igual a SAIR
       contadorRótulos = 0
       enquanto contadorRótulos não igual a RÓTULOS
          imprimir RÓTULO_TEXTO, nome
          contadorRótulos = contadorRótulos + 1
       fim-enquanto
       imprimir "Próximo nome ou " , SAIR, "para sair... "
       obter nome
    fim-enquanto
fim
```

Figura 5-8 Programa que produz cem rótulos com o nome de cada empregado (*continuação*)

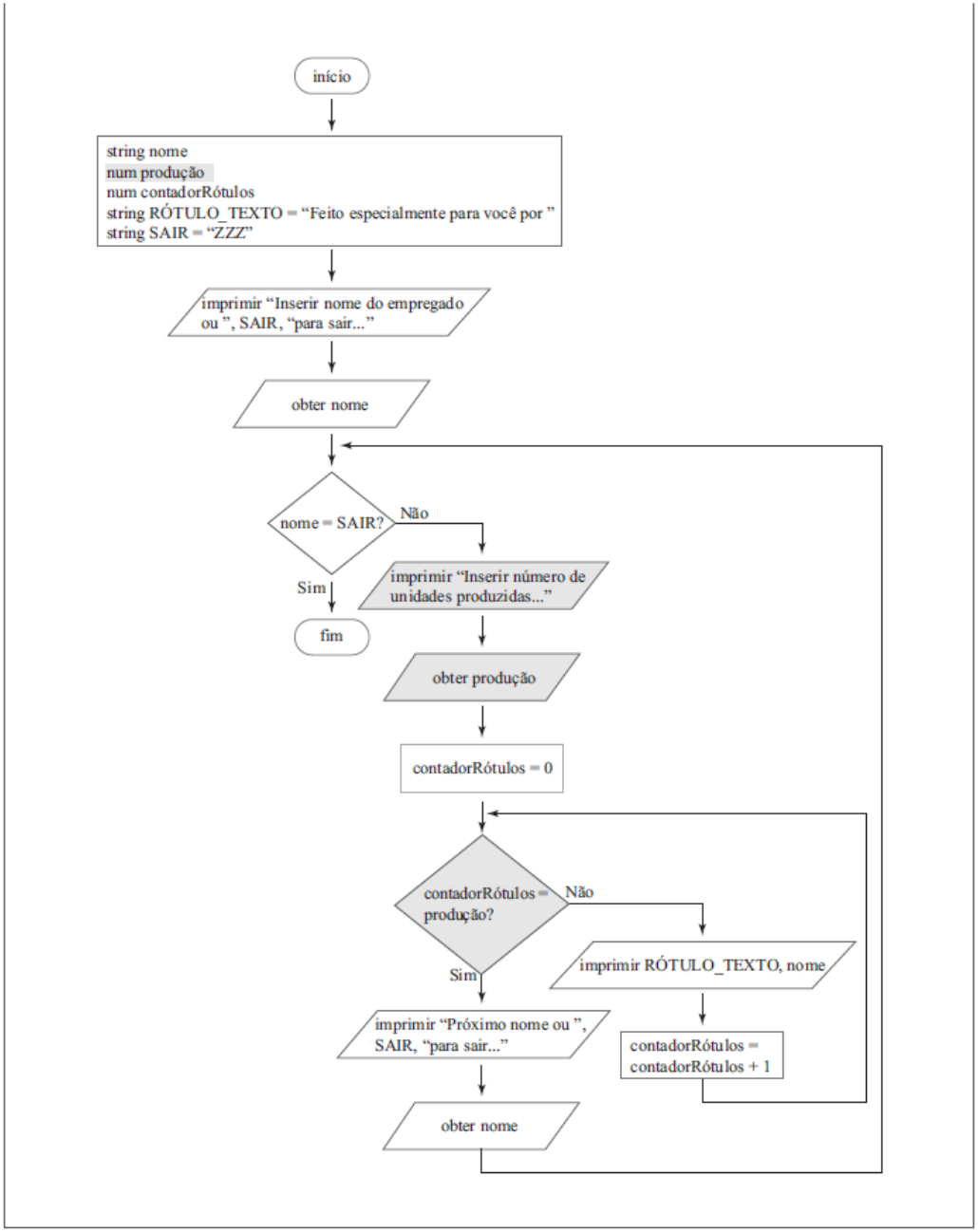


Figura 5-9 Programa que produz um número variável de rótulos para cada empregado (*continuação*)

Evitando erros comuns com loops

- Negligenciar a inicialização da variável de controle do loop
- Negligenciar a alteração da variável de controle do loop
- Usar a comparação errada com a variável de controle do loop
- Incluir sentenças dentro do loop que pertenceriam à sua parte externa

Evitando erros comuns com loops (cont.)

- Erro: negligenciar a inicialização da variável de controle do loop
  - Exemplo: sentença `obter nome` removida
    - Valor do `nome` é desconhecido, ou seja, lixo
    - O programa pode finalizar antes que qualquer rótulo pudesse ser impresso
    - Se entrasse no loop interno e 100 rótulos seriam impressos com um nome inválido
  - Sentença `contadorRótulos = 0` removida
    - Valor de `contadorRótulos` imprevisível
    - O loop pode executar ou não
    - Variáveis numéricas são automaticamente inicializadas em 0, apenas os primeiros rótulos são impressos

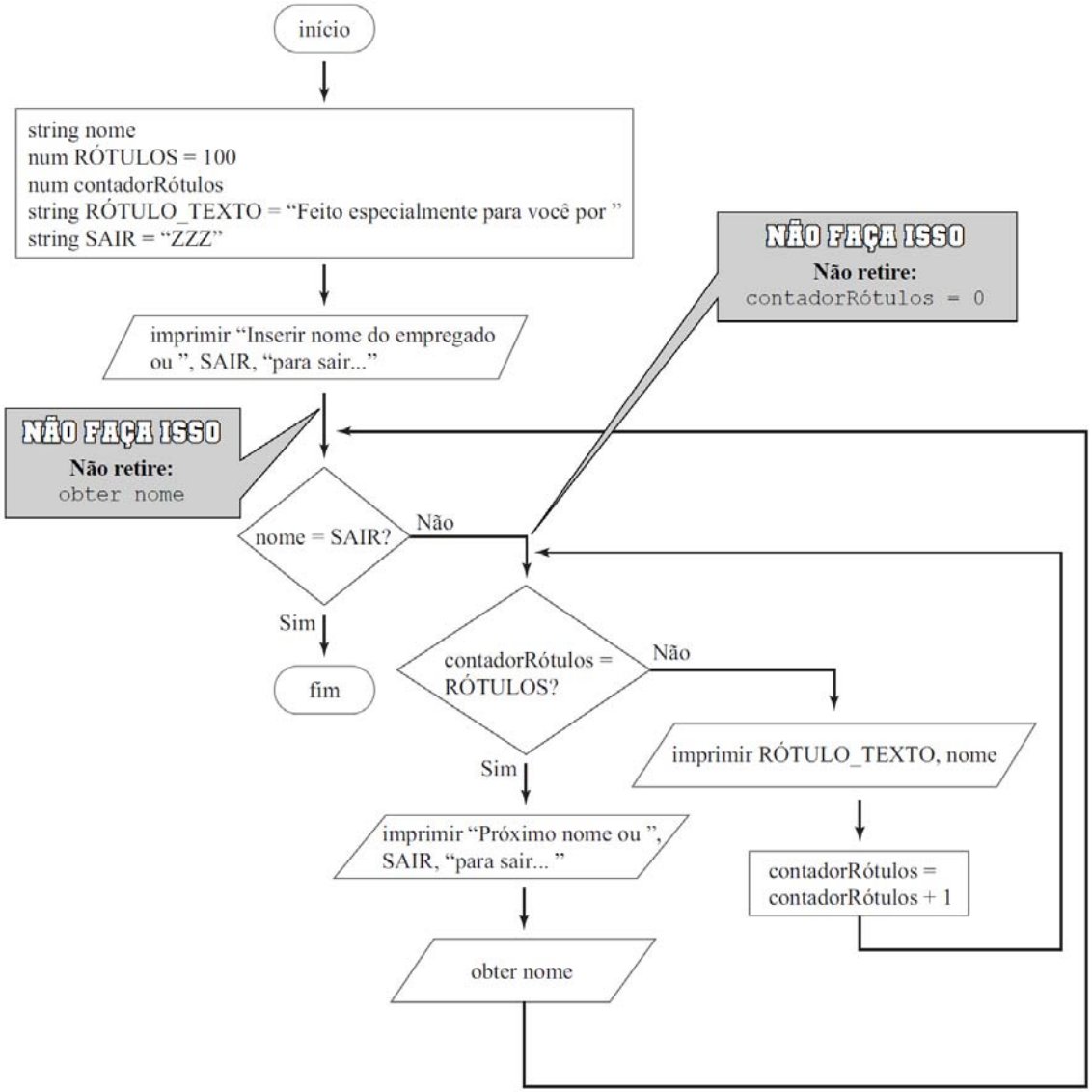
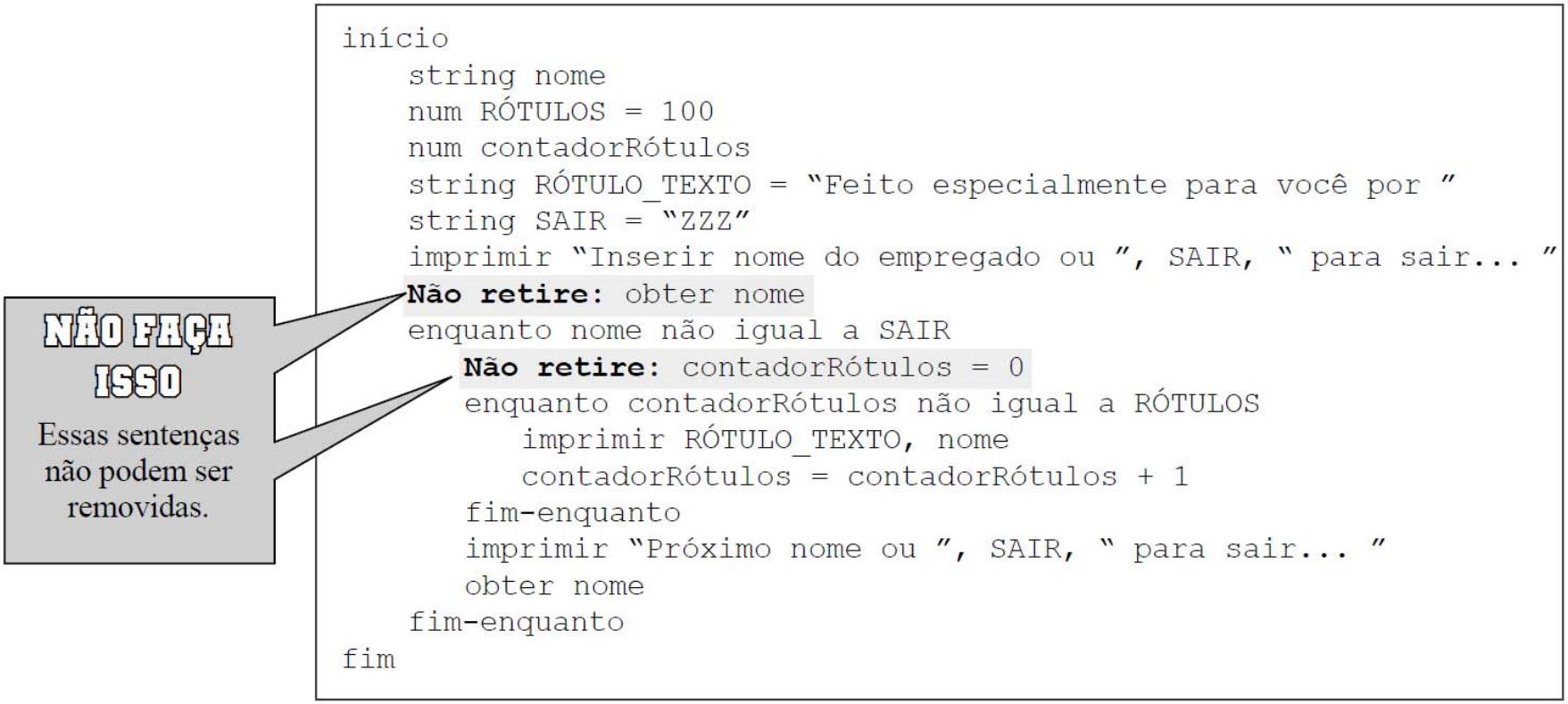


Figura 5-10 Lógica incorreta para o programa de rótulos, em que as inicializações da variável de controle do loop foram omitidas

Evitando erros comuns com loops (cont.)




```
início
    string nome
    num RÓTULOS = 100
    num contadorRótulos
    string RÓTULO_TEXTO = "Feito especialmente para você por "
    string SAIR = "ZZZ"
    imprimir "Inserir nome do empregado ou ", SAIR, " para sair... "
    Não retire: obter nome
    enquanto nome não igual a SAIR
        Não retire: contadorRótulos = 0
        enquanto contadorRótulos não igual a RÓTULOS
            imprimir RÓTULO_TEXTO, nome
            contadorRótulos = contadorRótulos + 1
        fim-enquanto
        imprimir "Próximo nome ou ", SAIR, " para sair... "
        obter nome
    fim-enquanto
fim
```

Figura 5-10 Lógica incorreta para o programa de rótulos, em que as inicializações da variável de controle do loop foram omitidas (cont.)

Evitando erros comuns com loops (cont.)

- Erro: negligenciar a alteração da variável de controle do loop
  - Remoção da instrução `obter nome` do loop externo do programa
  - O usuário nunca terá a oportunidade de inserir um nome depois do primeiro
  - Loop interno executa infinitamente
  - Remocão da sentença que incrementa `contadorRótulos` do loop interno
    - O loop interno executará infinitamente
  - Sempre incorreto criar um loop que não pode terminar

Evitando erros comuns com loops (cont.)

- Erro: usar a comparação errada com a variável de controle do loop
  - Programadores precisam ser cuidadosos e usar a comparação correta
  - A gravidade do erro gerado por usar <= ou >= quando apenas < ou > seriam necessários depende da ação realizada no loop
  - A gravidade depende de ações realizadas dentro de um loop
    - Valor mensal adicional de juros de assegurados
    - *Overbooking* de um voo
    - Dispensar medicamento extra para pacientes em farmácia

Evitando erros comuns com loops (cont.)

- Exemplo:
  - Correto: loop executado 10 vezes

```
contador = 0
enquanto contador < 10
   imprimir “Olá”
   contador = contador + 1
fim-enquanto
```

Evitando erros comuns com loops (cont.)

- Exemplo (cont.):
  - Errado: loop executado 11 vezes

```
contador = 0
enquanto contador <= 10
   imprimir “Olá”
   contador = contador + 1
fim-enquanto
```

Evitando erros comuns com loops (cont.)

- Erro: incluir sentenças dentro do loop que pertenceriam à sua parte externa
  - Exemplo: cálculo do aumento projetado para o pagamento semanal de um usuário
  - O valor do pagamento semanal é recalculado em cada passagem pelo loop, apesar deste não mudar
    - Ineficiente, especialmente para cálculos complicados ou grande número de cálculos

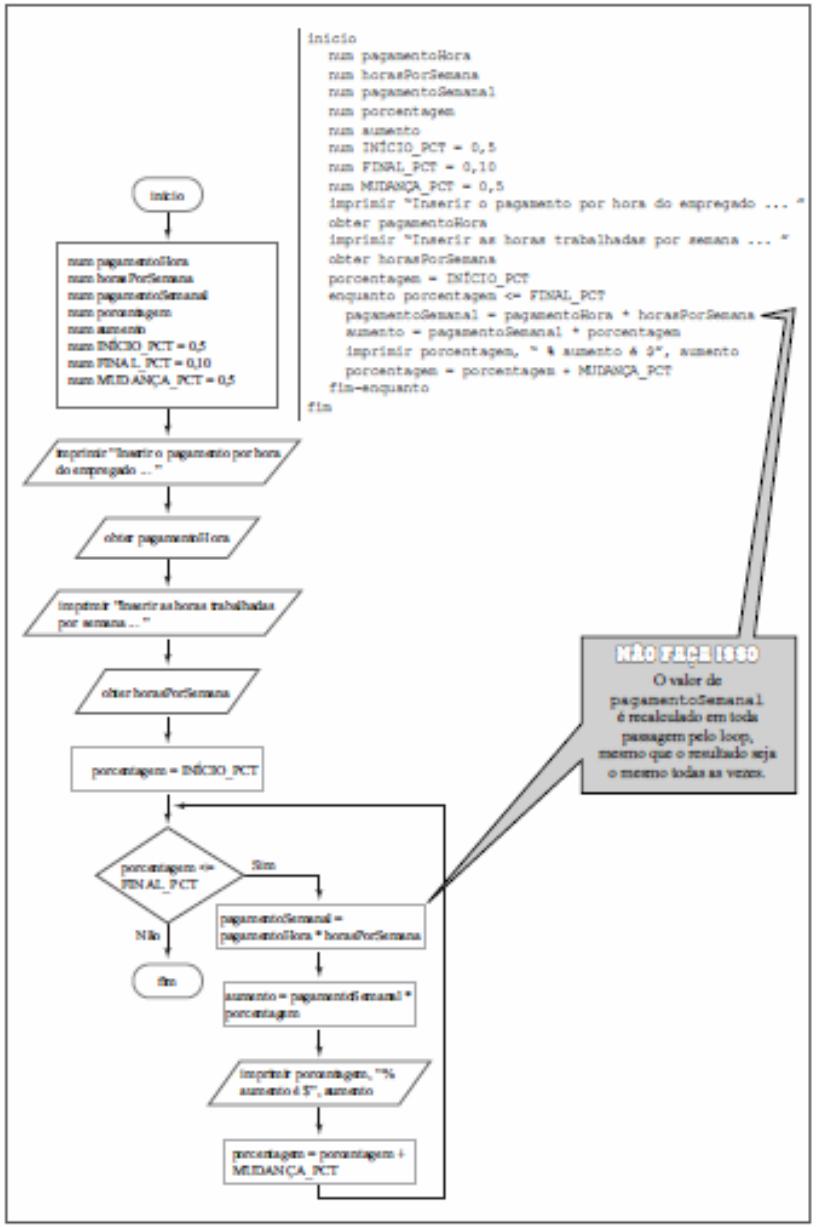


Figura 5-12 Programa de projeção do valor de pagamento semanal

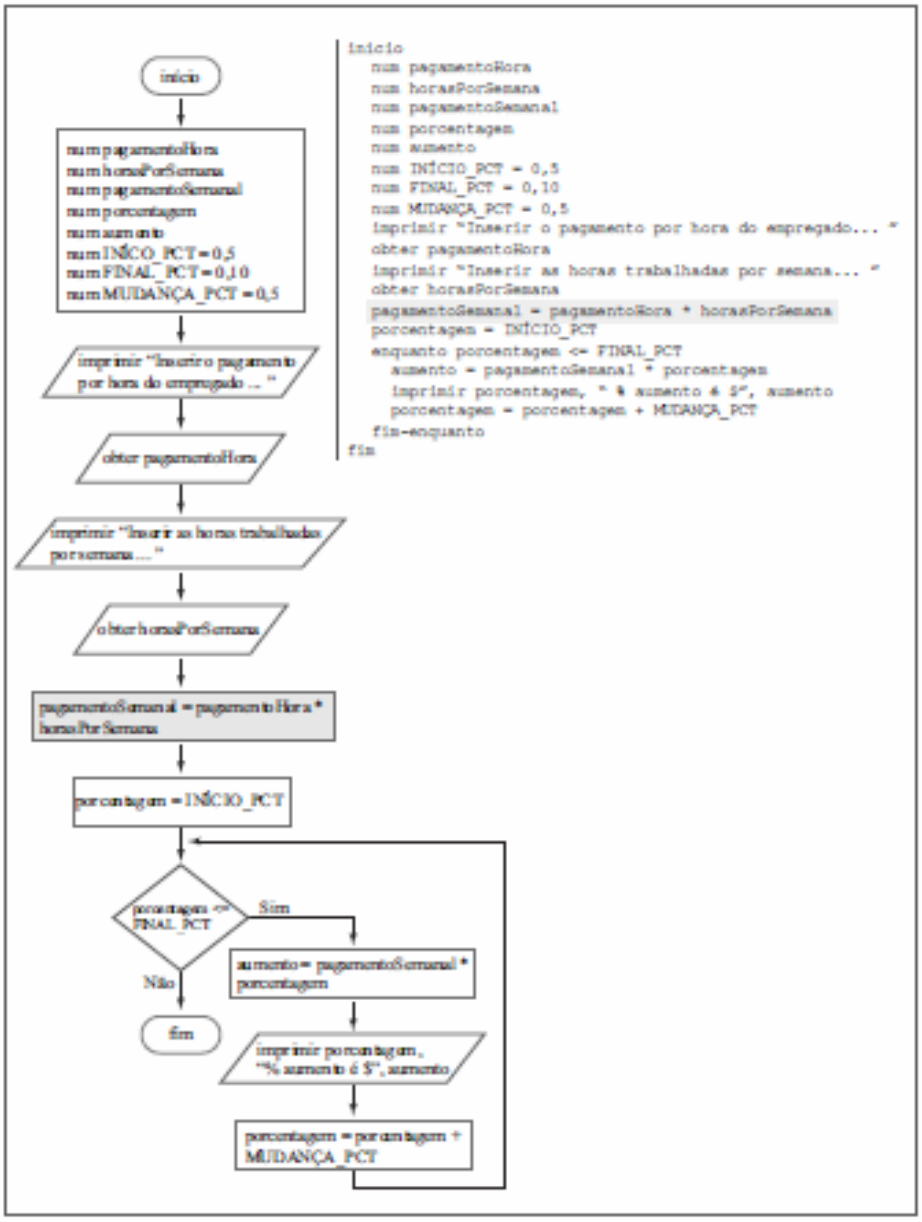


Figura 5-14 Programa melhorado de projeção do valor de pagamento semanal

Usando um loop `for`

- **Sentença `for`,** ou **loop `for`**, com loops definidos
- Uma variável de controle do loop usada por uma sentença for faz as seguintes ações automaticamente:
  - Inicialização
  - Verificação
  - Incrementação
- A sentença **for** tem o formato:

```
for valorInicial to valorFinal
    fazer algumaCoisa
fim-for
```

Usando um loop `for` (cont.)

- Exemplo:

```
for contador = 0 to 99
   imprimir RÓTULO_TEXTO, nome
fim-for
```

- A sentença `for` inicializa `contador` em `0`
- A sentença `for` testa `contador` contra o valor-limite de 99
- Se a avaliação for verdadeira, o corpo da sentença `for` que imprime os rótulos é executado
- O valor de `contador` aumenta em 1

Usando um loop `for` (cont.)

- A sentença `enquanto` poderia ser usada no lugar da sentença `for`
- **Valor de etapa:** um número usado para aumentar a variável de controle do loop a cada passagem por ele
  - Em algumas linguagens, é preciso sempre fornecer uma sentença que indique o valor de etapa do loop `for`
  - Em outras, o valor de etapa default dos loops é 1
- Especifica-se um valor de etapa quando quer que cada passagem pelo loop altere a variável de controle dele por um valor diferente de 1

Usando loops pós-teste

- Quando se usa um loop **enquanto** ou um loop **for**, pode acontecer de o corpo do loop nunca executar
- Quando se quiser garantir que o corpo de um loop execute ao menos uma vez, pode-se usar um loop pós-teste
  - Loop `fazer-até`
  - Loop `fazerenquanto`
- Loop `fazer-até` é executado até que determinada condição torne-se verdadeira
- `Loop fazereenquanto` é executado enquanto determinada condição permanecer verdadeira

Usando loops pós-teste (cont.)

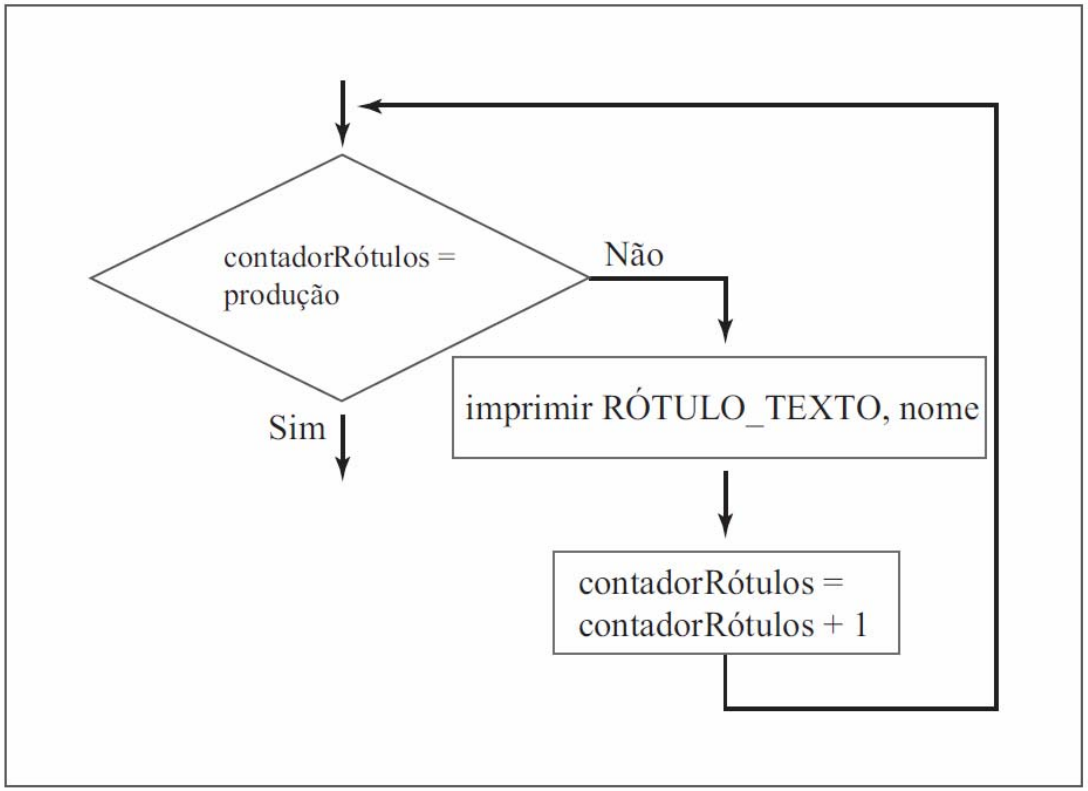


Figura 5-15 Loop interno do programa de produção de rótulos da Figura 5-9

Usando loops pós-teste (cont.)

```
início
    string nome
    num produção
    num contadorRótulos
    string RÓTULO_TEXTO = "Feito especialmente para você por "
    string SAIR = "ZZZ"
    imprimir "Inserir nome do empregado ou ", SAIR, "para sair... "
    obter nome
    enquanto nome não = SAIR
       imprimir "Inserir número de unidades produzidas... "
       obter produção
       contadorRótulos = -1
       fazer
          imprimir RÓTULO_TEXTO, nome
          contadorRótulos = contadorRótulos + 1
       até contadorRótulos = produção
       imprimir "Próximo nome ou ", SAIR, "para sair... "
       obter nome
    fim-enquanto
```

Figura 5-16 Programa de produção de rótulos usando um loop fazer-até

Reconhecendo as características compartilhadas por todos os loops

- Qualquer problema de lógica poderia ser resolvido usando apenas o loop `enquantoEvery`
  - Outras formas são úteis em certas situações
- Loop `enquanto`
  - A questão de controle do loop é posta no começo dos passos que se repetem
- Loop `fazer-até`
  - A questão de controle do loop é colocada no final da sequência de passos que se repetem

Reconhecendo as características compartilhadas por todos os loops (cont.)

- Características de todo loop estruturado:
  - A questão de controle do loop precisa fornecer uma entrada ou uma saída da estrutura de repetição
  - A questão de controle do loop fornece a *única* entrada ou saída da estrutura de repetiçã
- Há exatamente um valor de controle do loop e ele provê a única entrada ou a única saída do loop

Reconhecendo as características compartilhadas por todos os loops (cont.)

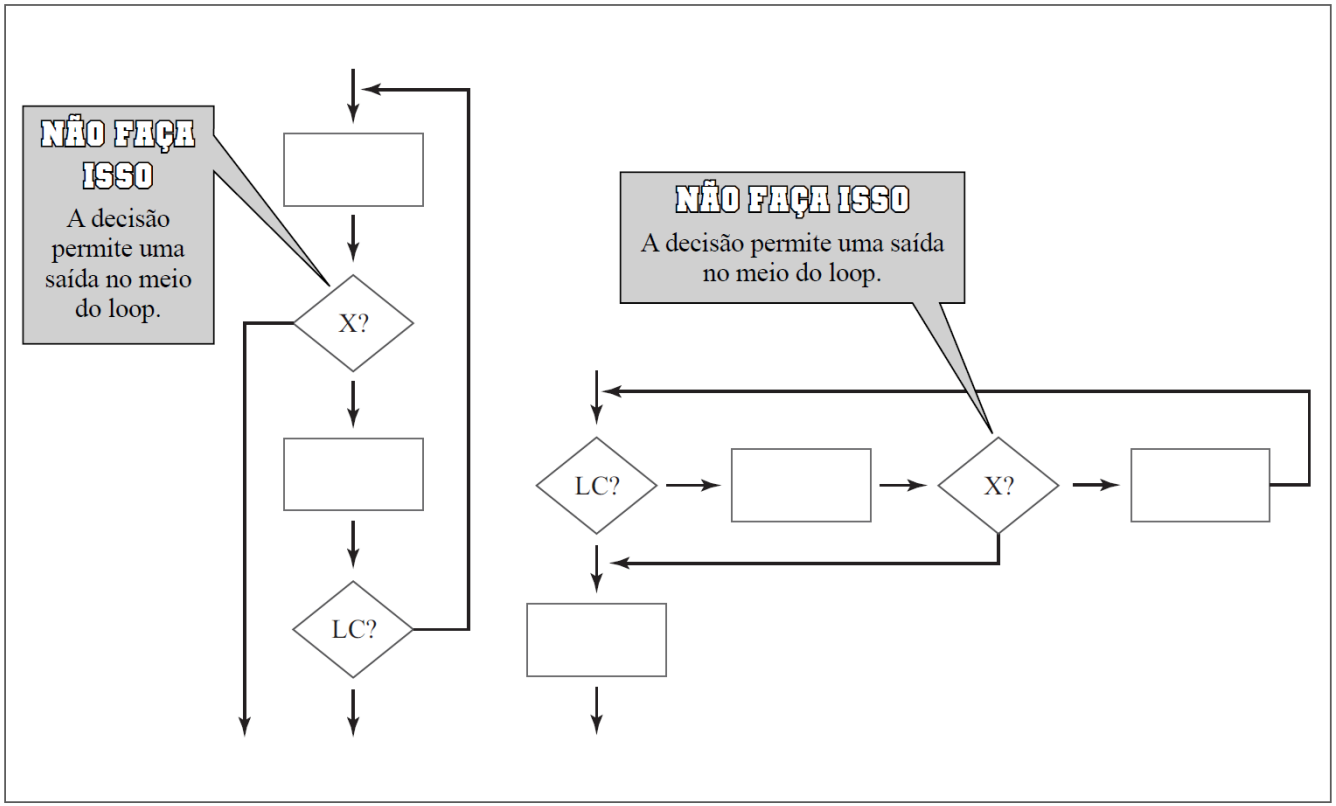


Figura 5-17 Exemplos de loops desestruturados

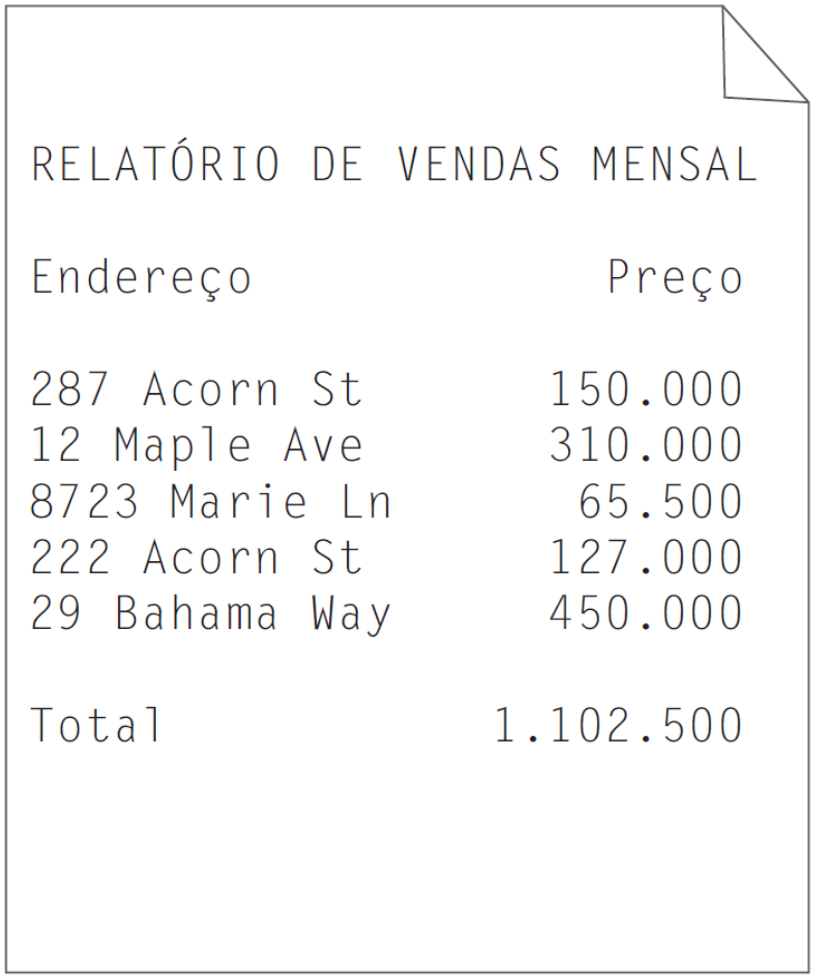
Aplicações comuns de loops

- Usando um loop para acumular totais
  - Exemplos:
    - Relatórios de negócios muitas vezes incluem totais
    - Contas de telefone geram um total
    - Um corretor de imóveis pode querer ver uma lista com todas as propriedades vendidas no último mês
- **Acumulador:** uma variável que usada para reunir ou acumular valores e é muito parecido com um contador que você usa para contar iterações de loops

Aplicações comuns de loops (cont.)

- Para acumular os preços totais dos imóveis:
  - Declara-se uma variável numérica no começo da aplicação
  - O acumulador, acumValor, deve ser iniciado em 0
  - Lê-se os registros transações
  - O programa imprime os registros e adiciona seu valor ao acumulador acumValor
  - O próximo registro é lido até `eof`
- As variáveis existem apenas para a vida da aplicação
  - Se ocorrer de uma aplicação futura conter uma variável nomeada acumValor, a variável não vai necessariamente ocupar o mesmo local da memória que a outra

Aplicações comuns de loops (cont.)



```
RELATÓRIO DE VENDAS MENSAL

Endereço                Preço

287 Acorn St          150.000
12 Maple Ave          310.000
8723 Marie Ln          65.500
222 Acorn St          127.000
29 Bahama Way         450.000

Total                1.102.500
```

Figura 5-18 Relatório das vendas da imobiliária ao final do mês

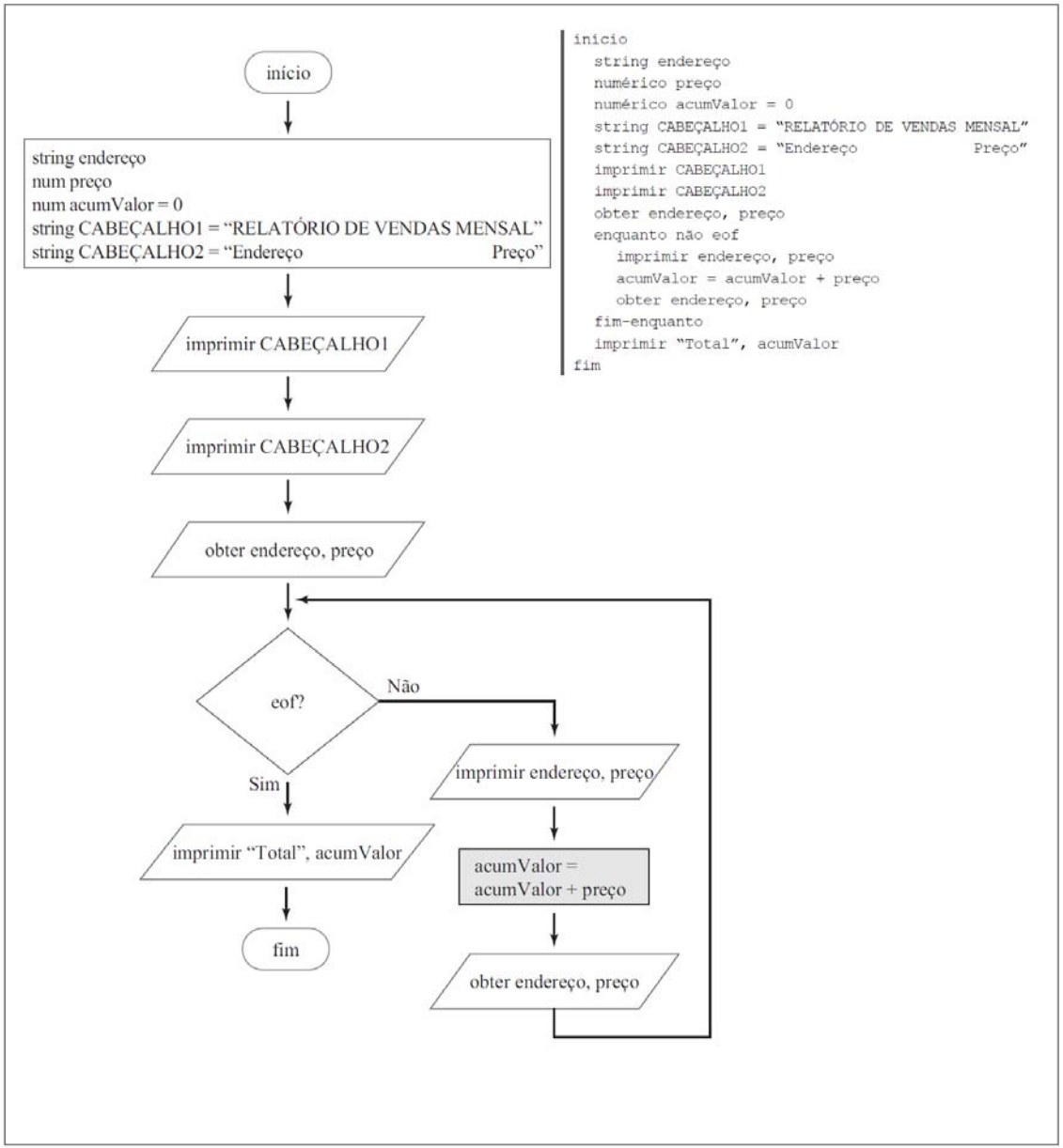


Figura 5-19 Fluxograma e pseudocódigo para programa de relatório de vendas da imobiliária

Aplicações comuns de loops (cont.)

- Usando um loop para validar dados
  - Quando você pede para um usuário inserir dados no programa de computador, não há nenhuma garantia de que os dados que ele inseriu sejam precisos
- **Validar dados**: para se ter certeza de que estão dentro de faixas aceitáveis

Aplicações comuns de loops (cont.)

- Usando um loop para validar dados
- Exemplo: usuário entra com seu mês de nascimento
  - Se o número for menor que 1 ou maior que 12
    - Você poderia apresentar uma mensagem de erro e interromper o programa
    - Você poderia optar por designar um valor default para o mês, por exemplo 1, antes de proceder
    - Você poderia novamente solicitar que o usuário fizesse uma entrada válida

Aplicações comuns de loops (cont.)

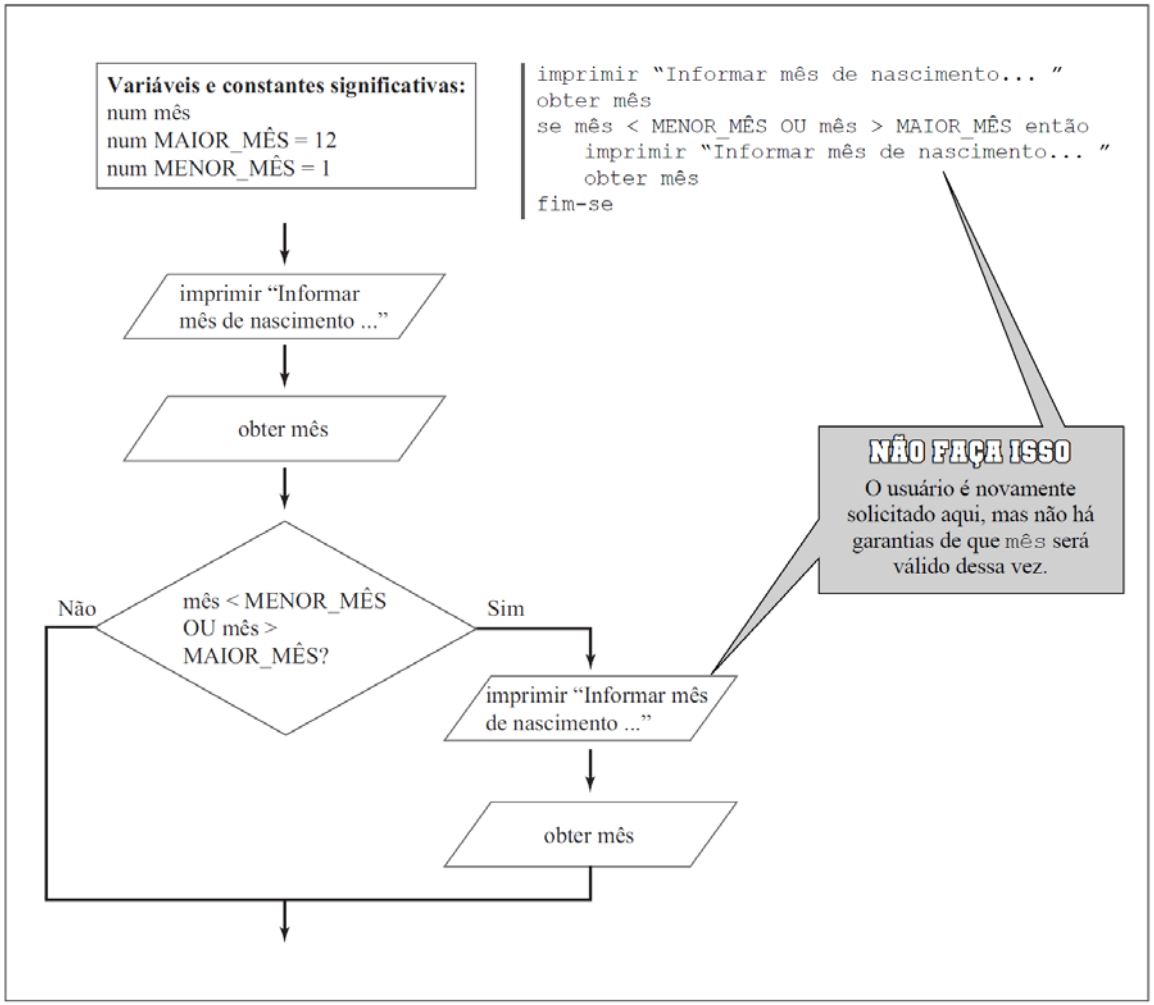


Figura 5-20 Solicitando novamente entrada de um usuário, uma vez que um mês inválido foi inserido

Aplicações comuns de loops (cont.)

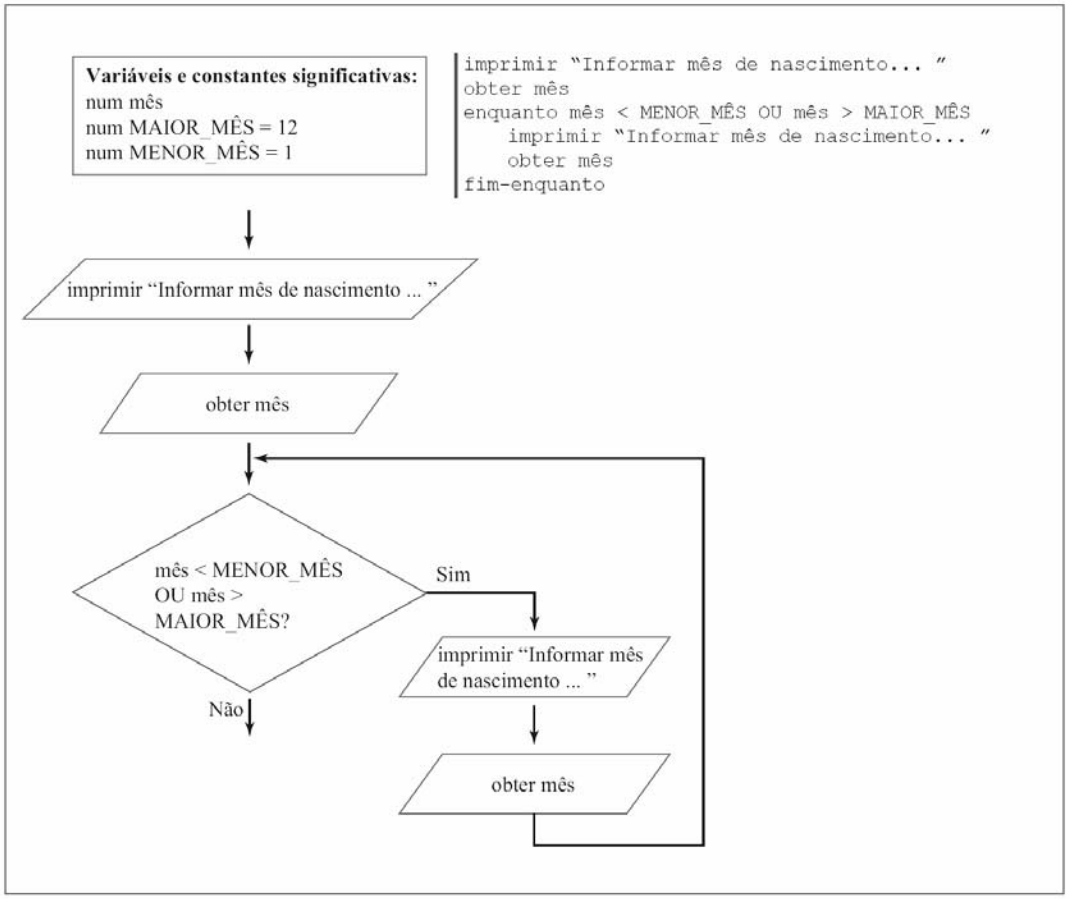


Figura 5-21 Solicitando continuamente um novo mês depois que um valor inválido foi inserido

Resumo do Capítulo

- Usando um loop dentro de um programa de computador, você pode escrever um conjunto de instruções para operar em múltiplos conjuntos separados de dados
- Três passos precisam ocorrer em qualquer loop:
  - A inicialização de uma variável de controle do loop
  - A comparação da variável a um valor de controle para verificar se o loop continua ou para
  - A alteração da variável de controle do loop:
- Loops embutidos: loops dentro de loops
- Ao embutir loops, é preciso manter duas variáveis de controle dos loop individuais
  - Alterar cada uma no momento apropriado

Resumo do Capítulo (cont.)

- Erros comuns que programadores cometem:
  - Negligenciar a inicialização da variável de controle do loop
  - Negligenciar a alteração da variável de controle do loop
  - Usar a comparação errada com a variável de controle do loops
  - Incluir sentenças dentro do loop que pertencem à parte externa do loop

Resumo do Capítulo (cont.)

- A maioria das linguagens de computador aceita uma sentença **for** ou **loop for**
- Esta pode ser usada com loops definidos
  - Quando souber quantas vezes esse loop vai se repetir
- A sentença `for` automaticamente:
  - Initializa
  - Avalia
  - Incrementa
- Quando se quer garantir que o corpo de um loop execute pelo menos uma vez, pode-se usar um loop pós-teste

Resumo do Capítulo (cont.)

- Características de todo loop estruturado:
  - A questão de controle do loop fornece entrada ou saída da estrutura de repetição
  - A questão de controle do loop provê a *única* entrada ou saída da estrutura de repetição
- Acumulador: uma variável usada para reunir ou acumular valores
- Loops também são usados para garantir que dados informados sejam válidos continuamente solicitando entradas do usuário