No Visual FoxPro, a programação procedural e a programação orientada a objetos trabalham juntas para que você possa criar aplicativos poderosos e flexíveis. Conceitualmente, você pode entender a programação como uma forma de escrever uma seqüência de instruções para executar tarefas específicas. Em um nível estrutural, a programação em Visual FoxPro envolve a manipulação de dados armazenados.

Se você não tiver experiência em programação, este capítulo o ajudará a dar os primeiros passos. Se você já estiver familiarizado com outras linguagens de programação e desejar fazer uma comparação com o Visual FoxPro, consulte Visual FoxPro e outras linguagens de programação. Para obter uma explicação sobre a programação orientada a objetos, consulte o capítulo 3, Programação orientada a objetos

Este capítulo abrange os tópicos a seguir:

- Vantagens da programação
- O mecanismo de programação em Visual FoxPro
- Conceitos básicos de programação
- O processo de programação
- Utilizando procedimentos e funções definidas pelo usuário
- Seguindo adiante

Vantagens da programação

Em geral, tudo o que você pode fazer em um programa pode também ser feito manualmente se você tiver tempo suficiente. Caso queira procurar informações sobre um único cliente em uma tabela de clientes, a empresa Ernst Handel, por exemplo, você poderá fazer isso manualmente, seguindo uma seqüência específica de instruções.

Para localizar manualmente um único pedido em uma tabela

1. No menu **Arquivo**, selecione **Abrir**.
2. Na caixa **Arquivos do tipo**, selecione **Tabela**.
3. Clique duas vezes em CUSTOMER.DBF na lista de arquivos.
4. No menu **Exibir**, selecione **Pesquisar**.
5. Percorra a tabela procurando “Ernst Handel” no campo Company dos registros.

Por programação, você poderia obter os mesmos resultados, digitando os comandos a seguir do Visual FoxPro na janela **Comando**:

```
USE Customer
LOCATE FOR Company = "Ernst Handel"
BROWSE
```

Uma vez localizado o pedido para esta empresa, talvez você deseje aumentar em 3% a quantidade máxima de pedidos.

Para aumentar manualmente a quantidade máxima de pedidos

1. Vá com a tecla Tab até o campo `max_ord_amt`.
2. Multiplique o valor existente em `max_ord_amt` por 1,03 e digite o novo valor no campo.

Para obter o mesmo resultado por programação, digite o comando a seguir do Visual FoxPro na janela **Comando**:

```
REPLACE max_ord_amt WITH max_ord_amt * 1.03
```

É relativamente simples alterar a quantidade máxima de pedidos de um único cliente, manualmente ou digitando as instruções na janela **Comando**. Suponha, no entanto, que você deseje aumentar em 3% a quantidade máxima de pedidos de todos os clientes. A mesma tarefa executada manualmente consome muito tempo e é propensa a inúmeros erros. Se você fornecer as instruções corretas em um arquivo de programa, o Visual FoxPro poderá executar essa tarefa de forma rápida e fácil, sem

erros.

Programa de exemplo para aumentar a quantidade máxima de pedidos para todos os clientes

Código	Comentários
`USE customer`	Abra a tabela CUSTOMER.
`SCAN`	Percorre todos os registros na tabela e executa todas as instruções entre SCAN e ENDSCAN para cada registro.
`REPLACE max_ord_amt WITH ;` `  max_ord_amt * 1.03`	Aumenta em 3% a quantidade máxima de pedidos. (O ponto-e-vírgula (;) indica que o comando continua na linha seguinte.)
`ENDSCAN`	Fim do código executado para cada registro na tabela.

É bem mais vantajoso executar um programa do que digitar comandos individuais na janela **Comando**:

- Os programas podem ser modificados e executados novamente.
- Você pode executar programas a partir de seus menus, formulários e barras de ferramentas.
- Os programas podem executar outros programas.

As seções a seguir detalham o mecanismo, os conceitos e os processos subjacentes a este e outros programas do Visual FoxPro.

O mecanismo de programação em Visual FoxPro

Você pode programar o Visual FoxPro escrevendo um código: instruções sob a forma de comandos, funções ou operações que o Visual FoxPro pode entender. Você pode incluir essas instruções nos locais a seguir:

- Na janela **Comando**
- Em arquivos de programa
- Em janelas de código de evento ou de método no **Criador de formulários** ou no **Criador de classes**
- Em janelas de código de procedimento no **Criador de menus**
- Em janelas de código de procedimentos no **Criador de relatórios**

Utilizando a janela Comando

Você pode executar um comando do Visual FoxPro, digitando-o na janela **Comando** e pressionando ENTER. Para executar o comando outra vez, mova o cursor para a linha que contém o comando e pressione ENTER novamente.

Você pode até executar várias linhas de código na janela **Comando** como se fossem um programa independente.

Para executar várias linhas de código na janela Comando

1. Selecione as linhas de código.
2. Pressione ENTER ou selecione **Executar seleção** no menu de atalho.

Como a janela **Comando** é uma janela de edição, você pode editar os comandos, utilizando as ferramentas de edição disponíveis no Visual FoxPro. Você pode editar, inserir, excluir, recortar, copiar ou colar texto na janela **Comando**.

A vantagem de digitar um código na janela **Comando** é que as instruções são executadas

imediatamente. Não é necessário salvar um arquivo e executá-lo como um programa.

Além disso, as seleções efetuadas nos menus e caixas de diálogo são repetidas na janela **Comando** como comandos. Você pode copiar e colar esses comandos em um programa do Visual FoxPro e depois executar o programa várias vezes, facilitando a execução de milhares de comandos repetidamente.

Criando programas

Um programa do Visual FoxPro é um arquivo de texto que contém uma série de comandos. Você pode criar um programa no Visual FoxPro de uma destas formas:

Para criar um programa

1. No **Gerenciador de projetos**, selecione **Programas** na guia **Código**.
2. Escolha **Novo**.

– Ou –

1. No menu **Arquivo**, selecione **Novo**.
2. Na caixa de diálogo **Novo**, selecione **Programa**.
3. Selecione **Novo arquivo**.

– Ou –

- Na janela **Comando**, digite:

```
MODIFY COMMAND
```

O Visual FoxPro abre uma nova janela denominada Programa1. Você agora pode digitar seu programa nessa janela.

Salvando programas

Depois de criar um programa, não se esqueça de salvá-lo.

Para salvar um programa

- No menu **Arquivo**, escolha **Salvar**.

Se você tentar fechar um programa que ainda não foi salvo, uma caixa de diálogo será aberta para que você possa salvar ou descartar as alterações efetuadas.

Se você salvar um programa criado no **Gerenciador de projetos**, o programa será adicionado ao projeto.

Se você salvar um programa que ainda não foi nomeado, a caixa de diálogo **Salvar como** será aberta para que você possa especificar um nome para o programa. Uma vez salvo o programa, você pode executá-lo ou modificá-lo.

Modificando programas

Após salvar seu programa, você poderá modificá-lo. Primeiro, abra o programa, de uma das formas a seguir:

Para abrir um programa

- Se o programa estiver contido em um projeto, selecione-o no **Gerenciador de projetos** e escolha **Modificar**.

  – Ou –

- No menu **Arquivo**, selecione **Abrir**. Será exibida uma caixa de diálogo com uma lista de arquivos disponíveis. Na lista **Arquivos do tipo**, escolha **Programa**. Na lista de arquivos, selecione o programa que você deseja modificar e, em seguida, escolha **Abrir**.

  – Ou –

- Na janela **Comando**, digite o nome do programa a ser modificado:
  `MODIFY COMMAND myprogrm`
  – Ou –
- Na janela **Comando**, digite:
  `MODIFY COMMAND ?`
  Na lista de arquivos, selecione o programa que você deseja modificar e, em seguida, escolha **Abrir**.

Depois de abrir o programa, você pode fazer alterações. Ao terminar de fazer as alterações, não se esqueça de salvar o programa.

Executando programas

Após criar um programa, você pode executá-lo.

Para executar um programa

- Se o programa estiver contido em um projeto, selecione-o no **Gerenciador de projetos** e escolha **Executar**.
  – Ou –
- No menu **Programa**, selecione **Executar**. Na lista de programas, selecione o programa que será executado e escolha **Executar**.
  – Ou –
- Na janela **Comando**, digite `DO` e o nome do programa a ser executado:
  `DO myprogrm`

Escrevendo código nas ferramentas de criação do Visual FoxPro

O **Criador de formulários**, o **Criador de classes** e o **Criador de menus** permitem que você integre facilmente o código do programa com a interface do usuário, para que o código apropriado seja executado em resposta às ações do usuário. O **Criador de relatórios** permite que você crie relatórios complexos e personalizados, integrando o código ao arquivo de relatório.

Para aproveitar toda a capacidade do Visual FoxPro, você deverá utilizar essas ferramentas de criação. Para obter maiores informações sobre o **Criador de relatórios**, consulte o capítulo 7, Criando relatórios e etiquetas, no *Guia do Usuário*. Para obter maiores informações sobre o **Criador de classes**, consulte o capítulo 3, Programação orientada a objetos, neste manual. Para obter maiores informações sobre o **Criador de formulários**, consulte o capítulo 9, Criando formulários, e sobre o **Criador de menus**, consulte o capítulo 11, Criando menus e barras de ferramentas

Conceitos básicos de programação

Ao programar, você armazena dados e manipula esses dados com uma série de instruções. Os dados e os recipientes de armazenamento de dados são a matéria-prima da programação. As ferramentas utilizadas para manipular essa matéria-prima são os comandos, as funções e os operadores.

Armazenando dados

Os dados com os quais você trabalha provavelmente incluem quantidades de tempo, itens financeiros e contábeis, além de datas, nomes, descrições etc. Cada fragmento de dado é de determinado tipo: pertence a uma categoria de dados que você manipula de modos semelhantes. Você poderia trabalhar diretamente com esses dados sem armazená-los, mas perderia grande parte da flexibilidade e potência do Visual FoxPro. O Visual FoxPro fornece diversos recipientes de armazenamento para otimizar a manipulação dos dados.

Tipos de dados

Os tipos de dados determinam como os dados são armazenados e utilizados. Você pode multiplicar dois números mas não pode multiplicar caracteres. Você pode imprimir caracteres em caixa alta, mas números não. Alguns dos tipos de dados do Visual FoxPro estão listados na tabela que se segue:

Tipos de dados

Tipo	Exemplos
Numérico	123 3,1415 – 7
Caractere	“Seqüência de Teste” “123” “01/01/95”
Lógico	.T. .F.
Data	{01/01/95}
DataHora	{01/01/95 12:30:00}

Recipientes de dados

Os recipientes de dados permitem que você execute as mesmas operações em diversos dados. Por exemplo, se você somar as horas trabalhadas por um funcionário, multiplicá-las pelo salário/hora e, em seguida, deduzir os impostos para determinar quanto o funcionário deve receber, terá que executar essas operações para cada funcionário, a cada pagamento. Se armazenar essas informações em recipientes e executar as operações nos recipientes, você terá apenas que substituir os dados antigos pelos novos e executar o mesmo programa novamente. A tabela a seguir lista alguns dos principais recipientes para dados no Visual FoxPro:

Recipientes de dados

Tipo	Descrição
Variáveis	Elementos individuais de dados armazenados na RAM (Memória de Acesso Aleatório) de seu computador.
Registrosdetabelas	Várias linhas de campos predeterminados, cada uma podendo conter um dado predefinido. As tabelas são salvas no disco.
Matrizes	Diversos elementos de dados armazenados na RAM.

Manipulando dados

Os recipientes e os tipos de dados fornecem a você os blocos de construção necessários para manipular os dados As peças restantes são os operadores, as funções e os comandos.

Utilizando operadores

Os operadores reúnem os dados. Eis aqui os operadores mais comuns no Visual FoxPro.

Operadores

Operador	Tipos de dados válidos	Exemplo	Resultado
=	Todos	`? n = 7`	Imprime .T. se o valor armazenado na variável `n` for 7;

			caso contrário, imprime .F.
+	Numérico, Caractere, Data, DataHora	`? "Fox" + "Pro"`	Imprime “FoxPro”
! ou `NOT`	Lógico	`? !.T.`	Imprime .F.
*, /	Numérico	`? 5 * 5` `? 25 / 5`	Imprime 25 Imprime 5

Observação Um ponto de interrogação (?) antes de uma expressão faz com que um caractere de nova linha e os resultados da expressão sejam exibidos na janela de saída ativa, que geralmente é a janela principal do Visual FoxPro.

Lembre-se de que você deve utilizar sempre o mesmo tipo de dados ao utilizar um operador. As instruções a seguir armazenam dois dados numéricos em duas variáveis. Os nomes das variáveis começam com `n` para indicar que contêm dados numéricos, mas você poderia nomeá-las com qualquer combinação de caracteres alfanuméricos e sublinhados.

```
nPrimeiro = 123
nSegundo = 45
```

As instruções a seguir armazenam dois dados de caractere em duas variáveis. Os nomes de variáveis começam com `c` para indicar que contêm dados de caractere.

```
cPrimeiro = "123"
cSegundo = "45"
```

As duas operações a seguir, adição e concatenação, produzem resultados diferentes porque os tipos de dados contidos nas variáveis são diferentes.

```
? nPrimeiro + nSegundo
? cPrimeiro + cSegundo
```

Saída

```
168
12345
```

Como `cPrimeiro` é um dado de caractere e `nSegundo` é um dado numérico, você obterá um erro de não-correspondência de tipo de dado ao tentar emitir o comando a seguir:

```
? cPrimeiro + nSegundo
```

Você pode evitar esse problema, utilizando funções de conversão. Por exemplo, STR( ) retornará o caractere equivalente a um valor numérico e VAL( ) retornará o valor numérico equivalente a uma seqüência de caracteres de números. Essas funções e LTRIM( ), que remove os espaços à esquerda, permitem que você execute as operações a seguir:

```
? cPrimeiro + LTRIM(STR(nSegundo))
? VAL(cPrimeiro) + nSegundo
```

Saída

```
12345
168
```

Utilizando funções

As funções retornam um tipo específico de dados. Por exemplo, as funções STR( ) e VAL( ), utilizadas na seção anterior, retornam valores de caractere e numéricos, respectivamente. Como acontece com todas as funções, esses tipos de retorno estão documentados juntamente com as funções.

Existem cinco formas de chamar uma função do Visual FoxPro:

- Atribuir o valor de retorno da função a uma variável. A linha de código a seguir armazena a data atual do sistema em uma variável denominada `dHoje`:

```
dHoje = DATE( )
```

- Incluir a chamada da função em um comando do Visual FoxPro. O comando a seguir define o diretório padrão como o valor retornado pela função GETDIR( ):

```
CD GETDIR( )
```

- Imprimir o valor de retorno na janela de saída ativa. A linha de código a seguir imprime a hora atual do sistema na janela de saída ativa:

```
? TIME( )
```

- Chamar a função sem armazenar o valor de retorno em lugar nenhum. A chamada de função a seguir desativa o cursor:

```
SYS(2002)
```

- Incorporar a função a outra função. A linha de código a seguir imprime o dia da semana:

```
? DOW(DATE( ))
```

Outros exemplos de funções utilizadas neste capítulo são:

Função	Descrição
ISDIGIT( )	Retornará verdadeiro (.T.) se o caractere mais à esquerda em uma seqüência for um número; caso contrário, retornará falso (.F.).
FIELD( )	Retornará o nome de um campo.
LEN( )	Retornará o número de caracteres em uma expressão de caracteres.
RECCOUNT( )	Retornará o número de registros na tabela ativa no momento.
SUBSTR( )	Retornará o número de caracteres especificado de uma seqüência de caracteres, começando em um local especificado na seqüência.

Utilizando comandos

Um comando faz com que uma determinada ação seja executada. Cada comando possui uma sintaxe específica que indica o que deve ser incluído para que o comando funcione. Há também cláusulas opcionais associadas aos comandos que lhe permitem especificar em maiores detalhes o que você deseja.

Por exemplo, o comando USE permite que você abra e feche tabelas:

Sintaxe USE	Descrição
`USE`	Fecha a tabela na área de trabalho atual.
`USE customer`	Abre a tabela CUSTOMER na área de trabalho atual, fechando outra tabela já aberta na área de trabalho.
`USE customer IN 0`	Abre a tabela CUSTOMER na próxima área de trabalho disponível.
`USE customer IN 0 ;` `  ALIAS mycust`	Abre a tabela CUSTOMER na próxima área de trabalho disponível e atribui à área de trabalho o alias `mycust`.

Alguns exemplos de comandos utilizados neste capítulo são:

Comando	Descrição
DELETE	Marca registros especificados em uma tabela para exclusão.
REPLACE	Substitui o valor armazenado no campo do registro por

	um novo valor.
GO	Posiciona o ponteiro do registro em um local específico na tabela.

Controlando o fluxo de programas

O Visual FoxPro inclui uma categoria especial de comandos que “agrupam” outros comandos e funções, determinando quando e com que freqüência esses outros comandos e funções são executados. Esses comandos permitem *desvios condicionais* e *loops*, duas ferramentas de programação muitoeficazes. O programa a seguir ilustra desvios condicionais e loops. Esses conceitos estão descritos com detalhes após o exemplo.

Suponha que você tenha 10.000 funcionários e deseje dar um aumento de 3% àqueles com salário anual igual ou superior a R$ 30.000,00 e um aumento de 6% àqueles com salário anual abaixo de R$ 30.000,00. O programa de exemplo a seguir efetua essa tarefa.

Este programa pressupõe que uma tabela com um campo numérico chamado `salary` esteja aberta na área de trabalho atual. Para obter informações sobre áreas de trabalho, consulte a seção “Utilizando diversas tabelas” no capítulo 7, Trabalhando com tabelas.

Programa de exemplo para aumentar o salário dos funcionários

Código	Comentários
`SCAN`	O código entre SCAN e ENDSCAN é executado tantas vezes quanto o número de registros existente na tabela. Cada vez que o código for executado, o ponteiro do registro se moverá para o próximo registro na tabela.

`    IF salary >= 30.000,00`
`     REPLACE salary WITH ;`
`       salary * 1.03`
	Para cada registro, se o salário for maior ou igual a 30.000, substitui este valor por um novo salário 3% mais alto.
	O ponto-e-vírgula (;) depois de WITH indica que o comando continuará na linha seguinte.

`    ELSE`
`     REPLACE salary WITH ;`
`       salary * 1.06`
	Para cada registro, se o salário não for maior ou igual a 30.000, substitui este valor por um novo salário 6% mais alto.

`    ENDIF` `ENDSCAN`	Fim da instrução condicional IF. Fim do código executado para cada registro na tabela.

Este exemplo utiliza comandos de loop e desvio condicional para controlar o fluxo do programa.

Recurso de desvio condicional

O recurso de desvio condicional permite que você teste condições e, dependendo dos resultados desse teste, efetue diferentes operações. Há dois comandos no Visual FoxPro que permitem desvios condicionais:

- IF ... ELSE ... ENDIF
- DO CASE ... ENDCASE

O código entre a instrução inicial e a instrução ENDIF ou ENDCASE será executado somente se uma condição lógica tiver um valor verdadeiro (.T.). No programa de exemplo, o comando IF é utilizado para distinguir entre dois estados: se o salário é ou não igual ou superior a R$ 30.000,00. A ação a ser tomada depende do estado.

No exemplo a seguir, se o valor armazenado na variável `nWaterTemp` for menor que 100, nenhuma ação será tomada:

```
* define uma variável lógica como verdadeira se uma condição for atendida.
IF nWaterTemp >= 100
   lBoiling = .T.
ENDIF
```

Observação Um asterisco no início de uma linha em um programa indica que essa linha é um comentário. Os comentários lembram o programador do que cada segmento de código deve executar, mas são ignorados pelo Visual FoxPro.

Se houver várias condições possíveis a serem verificadas, um bloco DO CASE ... ENDCASE poderá ser mais eficiente e mais fácil de ser controlado do que várias instruções IF.

Recurso de loop

O recurso de loop permite que você execute uma ou mais linhas de código tantas vezes quantas forem necessárias. Existem três comandos no Visual FoxPro que permitem loop:

- SCAN ... ENDSCAN
- FOR ... ENDFOR
- DO WHILE ... ENDDO

Utilize SCAN quando estiver executando uma seqüência de ações para cada registro em uma tabela, como no programa de exemplo descrito anteriormente. O loop SCAN permite que você escreva o código uma vez e que ele seja executado para cada registro à medida que o ponteiro do registro percorre a tabela.

Utilize FOR quando você souber quantas vezes a seção de código deve ser executada. Por exemplo, você sabe que existe um número específico de campos em uma tabela. Como a função FCOUNT( ) do Visual FoxPro retorna este número, você pode utilizar um loop FOR para imprimir os nomes de todos os campos existentes na tabela:

```
FOR nCnt = 1 TO FCOUNT( )
   ? FIELD(nCnt)
ENDFOR
```

Utilize DO WHILE quando desejar executar uma seção de código, enquanto uma determinada condição for atendida. Você pode não saber quantas vezes o código terá que ser executado, mas sabe quando sua execução deverá ser interrompida. Por exemplo, suponha que você tivesse uma tabela com os nomes e as iniciais das pessoas e desejasse utilizar as iniciais para procurar as pessoas. Você enfrentaria um problema na primeira vez que tentasse incluir uma pessoa que tivesse as mesmas iniciais de outra já existente na tabela.

Para resolver o problema, você poderia adicionar um número às iniciais. Por exemplo, o código de identificação de Marcos Sabóia seria MS. A próxima pessoa com as mesmas iniciais, Margareth Silva, seria MS1. Se você adicionasse depois Márcia Santiago à tabela, seu código de identificação seria MS2. Um loop DO WHILE permite que você localize o número correto para anexar às iniciais.

Programa de exemplo com DO WHILE para gerar uma ID exclusiva

Código	Comentários
`nHere = RECNO()`	Salva a posição do registro.
`CInitials = LEFT(firstname,1) + ; LEFT(lastname,1)`	Extrai as iniciais da pessoa a partir das primeiras letras dos campos `firstname` e `lastname`.
`nSuffix = 0`	Estabelece uma variável para guardar o número a ser adicionado ao final das iniciais da pessoa, caso necessário.
`LOCATE FOR person_id = cInitials`	Verifica se há outra pessoa na tabela com as mesmas iniciais.
`DO WHILE FOUND( )`	Se um outro registro na tabela possuir um valor de person_id igual a `cInitials`, a função

	FOUND( ) retornará verdadeiro (.T.) e o código do loop DO WHILE será executado. Se nenhuma correspondência for encontrada, a próxima linha do código a ser executada será a seguinte a ENDDO.

`NSuffix = nSuffix + 1`
`cInitials = ;`
`LEFT(cInitials,2);`
`+ ALLTRIM(STR(nSuffix))`
	Prepara um sufixo novo e o anexa ao final das iniciais.

`CONTINUE`	CONTINUE faz com que o último comando LOCATE seja avaliado novamente. O programa verifica se o novo valor em `cInitials` já existe no campo person_id de outro registro. Se já existir, FOUND( ) retornará .T. e o código do loop DO WHILE será executado novamente. Se o novo valor em `cInitials` for realmenteexclusivo, FOUND( ) retornará .F. e a execução do programa continuará com a linha de código após ENDDO.
`ENDDO`	Fim do loop DO WHILE.
`GOTO nHere` `REPLACE person_id WITH cInitials`	Retorna ao registro e armazena o código de identificação exclusivo no campo `person_id`.

Como não é possível saber de antemão quantas vezes você encontrará códigos de identificação correspondentes já em uso, você deve utilizar o loop DO WHILE.

O processo de programação

Uma vez compreendidos os conceitos básicos, a programação é um processo iterativo. Você passa pelas etapas muitas vezes, refinando o seu código conforme vai programando. Quando está começando, você testa com freqüência utilizando várias vezes o método de tentativa e erro. Quanto mais familiarizado você estiver com a linguagem, mais rapidamente poderá programar e realizar um maior número de testes preliminares mentalmente

As etapas básicas de programação incluem:

- Definição do problema.
- Decomposição do problema em pequenos elementos.
- Construção das partes.
- Teste e ajuste das partes.
- Junção das partes.
- Teste de todo o programa.

Eis aqui alguns pontos a serem lembrados ao iniciar:

- Delineie claramente o problema antes de tentar resolvê-lo. Caso contrário, você acabará tendo que fazer diversas alterações, jogar código fora, começar tudo de novo ou aceitar algo aquém do que realmente deseja.
- Decomponha o problema em etapas que possam ser gerenciadas em vez de tentar resolver o problema inteiro de uma vez.
- Teste e depure seções de código enquanto desenvolve o programa. Verifique se o código faz o que você quer. A depuração é o processo de localização e solução dos problemas que impedem que o código faça o que você deseja.
- Aprimore os dados e o seu armazenamento para facilitar a manipulação destes dados pelo código do programa. Na maioria das vezes, isso significa estruturar corretamente suas tabelas.

O restante desta seção descreve as etapas para a criação de um pequeno programa do Visual FoxPro.

Definindo o problema

Antes de poder solucionar um problema, é necessário formulá-lo claramente. Algumas vezes, ajustando a formulação do problema, você será capaz de enxergar mais e melhores opções para solucioná-lo.

Suponha que você tenha diversos dados de várias origens. Embora a maior parte dos dados seja rigorosamente numérica, alguns valores de dados contêm traços e espaços além de números. Você deseja remover todos os espaços e traços desses campos e salvar os dados numéricos.

Em vez de tentar remover os espaços e traços dos dados originais, você pode formular o objetivo do programa como:

Objetivo Substituir os valores existentes em um campo por outros valores que contenham tudo o que pertence aos valores originais com exceção dos espaços e traços.

Esta formulação evita a dificuldade de manipular uma seqüência de caracteres cujo comprimento está sempre se modificando à medida que você trabalha com ela.

Decompondo o problema

Como você tem que fornecer instruções específicas para o Visual FoxPro em termos de operações, comandos e funções, será necessário decompor o problema em pequenas etapas. A menor tarefa para esse problema seria examinar cada caractere da seqüência. Até que possa examinar um caractere individualmente, você não pode determinar se deseja salvá-lo ou não.

Uma vez examinado um caractere, verifique se ele é um traço ou um espaço. Nesse momento, você pode desejar aprimorar a definição do problema. O que acontecerá se depois você obtiver dados com parênteses de abertura e de fechamento? O que acontecerá caso queira se livrar dos símbolos de moeda, vírgulas e pontos? Quanto mais genérico for o código, menos trabalho você terá mais tarde; o objetivo é economizar trabalho. Eis aqui uma formulação do problema que lida com uma maior variedade de dados:

Objetivo aprimorado Substituir os valores existentes em um campo por outros que contenham apenas os caracteres numéricos dos valores originais.

Com esta formulação, você pode agora redefinir o problema em nível de caractere: se for numérico, salve o caractere; caso contrário, vá para o próximo caractere. Após criar uma seqüência que contenha apenas os elementos numéricos da seqüência inicial, você pode substituir a primeira seqüência e ir para o próximo registro até passar por todos os dados.

Em resumo, o problema é decomposto nas partes a seguir:

1. Examinar cada caractere.
2. Determinar se o caractere é numérico ou não.
3. Se for numérico, copiá-lo para a segunda seqüência.
4. Após passar por todos os caracteres na seqüência original, substituir a seqüência original pela seqüência com apenas caracteres numéricos.
5. Repetir essas etapas para todos os registros na tabela.

Construindo as partes

Quando já souber o que fazer, você poderá começar a formular as partes em termos de comandos, funções e operadores do Visual FoxPro. Se souber exatamente qual o comando ou a função de que precisa, você poderá consultar a **Ajuda** para verificar a sintaxe correta. Caso saiba o que deseja fazer mas desconheça os comandos ou as funções apropriadas, localize o tópico “Categorias da

linguagem” na **Ajuda**.

Visto que os comandos e funções serão utilizados para manipular dados, será necessário trabalhar com dados de teste, pois você deseja que os dados de teste assemelhem-se tanto quanto possível aos dados reais.

Para este exemplo, você pode armazenar uma seqüência de teste para uma variável digitando o comando a seguir na janela **Comando**:

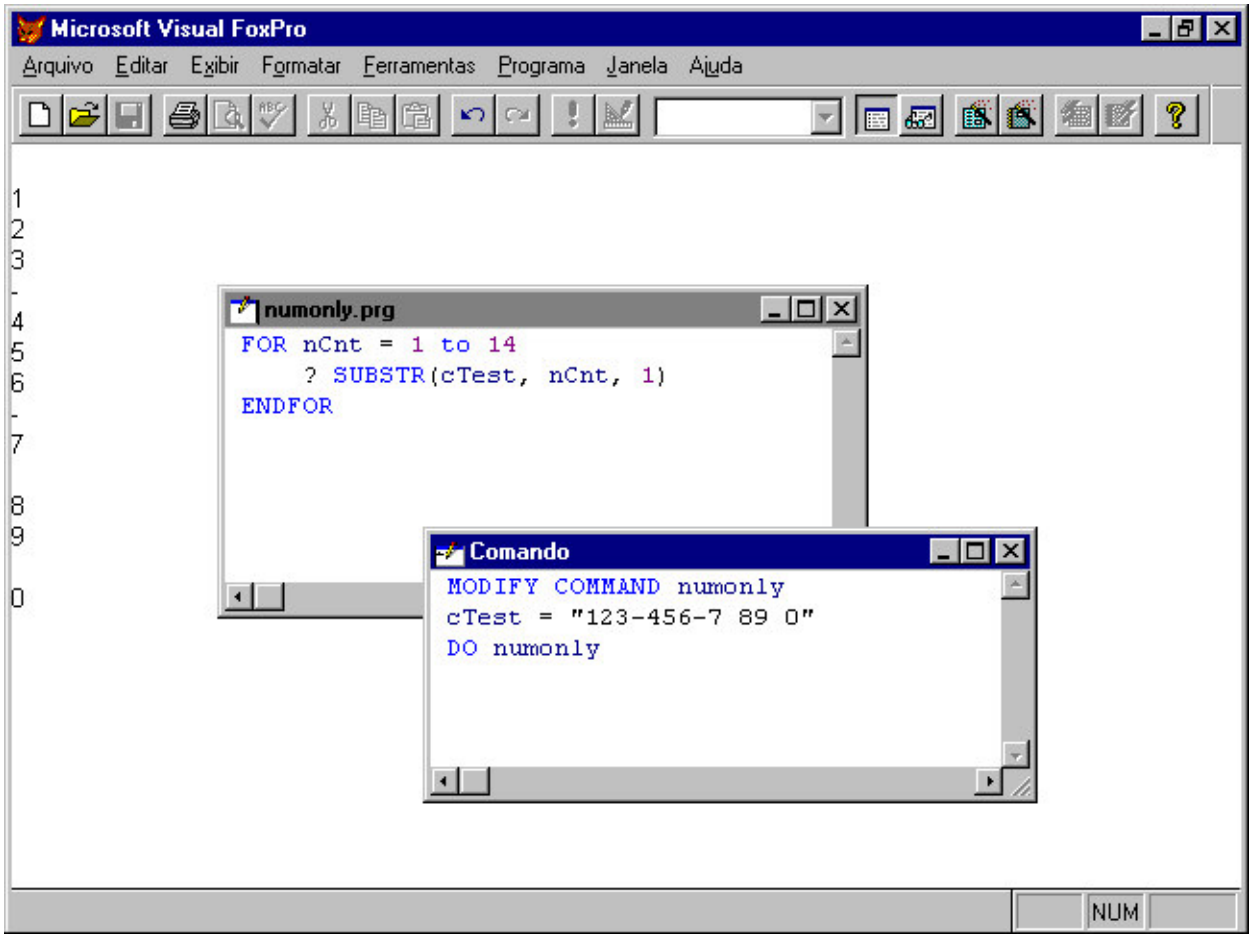
```
cTest = "123-456-7 89 0"
```

Examinando cada caractere

1. Primeiro, você deseja examinar um único caractere na seqüência. Consulte Funções de caracteres.
2. Se a caixa de diálogo **Salvar** for exibida, escolha **OK**.

Ao executar este programa, os caracteres individuais na seqüência de teste serão exibidos em linhas separadas na janela principal do Visual FoxPro.

Testando parte do programa



A primeira tarefa já foi realizada. Agora você pode examinar cada caractere na seqüência.

Decidindo se o caractere é numérico

Após isolar um único caractere da seqüência, será necessário saber se ele é um número. Você pode descobrir isso, utilizando ISDIGIT( ).

Digite os comandos a seguir na janela **Comando**:

```
? ISDIGIT('2')
? ISDIGIT('-')
? ISDIGIT(SUBSTR(cTest, 3, 1))
```

Saída

```
.T.
.F.
.T.
```

A partir dessa saída, você pode constatar que '2' é um número, '– ' não é um número e o terceiro caractere em `cTest`, 3, é um número.

Se o caractere for numérico, copie-o para a segunda seqüência

Agora que já pode examinar todos os caracteres e determinar se eles são numéricos ou não, você precisará de uma variável para guardar os valores numéricos: `cNumOnly`.

Para criar a variável, atribua a ela um valor inicial, uma seqüência de comprimento zero:

```
cNumOnly = ""
```

Como o loop FOR se move pela seqüência, é uma boa idéia criar outra variável para guardar temporariamente cada caractere da seqüência enquanto ela está sendo manipulada:

```
cCharacter = SUBSTR(cTest, nCnt, 1)
```

Dica Em geral, é melhor armazenar o resultado de um cálculo, avaliação ou função em uma variável. Você pode, então, manipular a variável sem precisar repetir o cálculo ou a avaliação.

A linha de código a seguir pode ser usada sempre que um número for encontrado para adicioná-lo à segunda seqüência:

```
cNumOnly = cNumOnly + cCharacter
```

Até agora, o programa inclui:

```
cNumOnly = ""
FOR nCnt = 1 TO 14
   cCharacter = SUBSTR(cTest, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
```

Testando as partes

Se adicionar alguns comandos no final para exibir as seqüências e, em seguida, executar o programa, você verá que o programa funciona com a seqüência de teste:

```
cNumOnly = ""
FOR nCnt = 1 TO 14
   cCharacter = SUBSTR(cTest, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
? cTest
? cNumOnly
```

Saída

```
123-456-7 89 0
1234567890
```

A saída parece correta. No entanto, se alterar a seqüência de teste enquanto estiver testando as partes, você poderá enfrentar problemas. Digite o comando a seguir na janela **Comando** e execute o programa novamente:

```
cTest = "456-789 22"
```
O programa irá gerar uma mensagem de erro. O loop FOR tentou executar 14 vezes mas há apenas 10 caracteres na seqüência. Você precisa de uma forma de ajuste para comprimentos de seqüência variáveis. Utilize LEN( ) para retornar o número de caracteres contido em uma seqüência. Se substituir este comando no loop FOR, você verá que o programa funciona corretamente com ambas as seqüências de teste:

```
cNumOnly = ""
FOR nCnt = 1 TO LEN(cTest)
   cCharacter = SUBSTR(cTest, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
? cTest
? cNumOnly
```

Juntando as partes

Para concluir a solução de programação para esse problema, convém que você leia seus dados a partir de uma tabela. Ao selecionar uma tabela a ser usada, varra os registros e aplique seu código de programa a um campo da tabela, e não a uma variável.

Primeiro, você pode criar uma tabela temporária com diversas seqüências de exemplo. Uma tabela dessa natureza pode conter um único campo de caractere denominado `TestField` e quatro ou cinco registros:

Conteúdo de TestField	
123-456-7 89 0	-9221 9220 94321 99-
456-789 22	000001 98-99-234

Ao substituir o nome do campo pelo nome da seqüência de teste, o programa ficará desta forma:

```
FOR nCnt = 1 TO LEN(TestField)
   cCharacter = SUBSTR(TestField, nCnt, 1)
   IF ISDIGIT(cCharacter)
      cNumOnly = cNumOnly + cCharacter
   ENDIF
ENDFOR
? TestField
? cNumOnly
```

Você pode ajustar manualmente o ponteiro do registro pesquisando e percorrendo a tabela. Quando o ponteiro está em cada registro, o programa funciona da forma desejada. Você pode também utilizar o código de navegação de tabela com o resto do programa:

```
SCAN
   cNumOnly = ""
   FOR nCnt = 1 TO LEN(TestField)
      cCharacter = SUBSTR(TestField, nCnt, 1)
      IF ISDIGIT(cCharacter)
         cNumOnly = cNumOnly + cCharacter
      ENDIF
   ENDFOR
? TestField
? cNumOnly
?
ENDSCAN
```

Saída

```
123-456-7 89 0
1234567890
```

```
456-789 22
45678922

 -9221 9220 94321 99-
922192209432199

000001 98-99-234
0000019899234
```

Testando todo o programa

Em vez de imprimir a seqüência no final do programa, você deseja salvá-la em sua tabela. Para isso, utilize a linha de código a seguir:

```
REPLACE TestField WITH cNumOnly
```

O programa completo agora é:

```
SCAN
   cNumOnly = ""
   FOR nCnt = 1 TO LEN(TestField)
      cCharacter = SUBSTR(TestField, nCnt, 1)
      IF ISDIGIT(cCharacter)
         cNumOnly = cNumOnly + cCharacter
      ENDIF
   ENDFOR
   REPLACE TestField WITH cNumOnly
ENDSCAN
```

Quando você tiver o programa completo, será necessário testá-lo com dados de exemplo antes de experimentá-lo em dados reais.

Tornando o programa mais eficaz

Um programa eficaz faz o que você deseja, mas também antecipa e lida com possíveis aspectos que poderiam dar errado. Este programa faz o que você deseja, mas faz algumas suposições que devem ser verdadeiras para que o programa funcione:

- Uma tabela está aberta na área de trabalho ativa.
- A tabela possui um campo de caractere denominado `TestField`.

Se a tabela não estiver aberta na área de trabalho ativa ou se ela não possuir um campo de caractere com o nome esperado, o programa irá gerar uma mensagem de erro e não conseguirá realizar a tarefa.

Programa para remover os caracteres não-numéricos de todos os registros em um campo

Código	Comentários
`lFieldOK = .F.`	Esta variável determina se existem as condições necessárias para que o programa funcione. Inicialmente, define a variável como falsa (.F.) para pressupor que as condições necessárias não existem.

`FOR nCnt = 1 TO FCOUNT( )`
`  IF FIELD(nCnt) = ;`
`  UPPER("TestField")`
`  IF TYPE("TestField") = "C"`
`  lFieldOK = .T.`
`  ENDIF`
`  EXIT`
`  ENDIF`
	Esta seção de código passa por cada campo na tabela ativa até encontrar um campo de caractere denominado `TestField`. Assim que o campo correto for localizado, `lFieldOK` será definida com verdadeira (.T.) e EXIT terminará o loop (não há mais razão para continuar a verificação uma vez que o campo correto já foi identificado). Se nenhum

	campo corresponder aos critérios, `lFieldOK` permanecerá como falsa (.F.).
`IF lFieldOK`	A seção de conversão do programa será executada apenas se um campo de caractere denominado `TestField` estiver presente na tabela ativa no momento.

`SCAN`
`  cNumOnly = ""`
`  FOR nCnt = 1 TO LEN(TestField)`
`  cCharacter = ;`
`  SUBSTR(TestField, nCnt, 1)`
`  IF ISDIGIT(cCharacter)`
`  cNumOnly = cNumOnly + ;`
`  cCharacter`
`  ENDIF`
`  ENDFOR`
`  REPLACE TestField WITH ;`
`  cNumOnly`
`ENDSCAN`
	O código de conversão.

`ENDIF`	Fim da condição de `IF lFieldOK`.

A característica de maior limitação deste programa é que você só pode usá-lo para um único campo. Se você desejar remover os caracteres não-numéricos de um campo que não seja `TestField`, terá que percorrer todo o programa e substituir cada ocorrência de `TestField` pelo nome do outro campo.

A conversão do programa em uma função, conforme será explicado nas seções a seguir, permite que você torne o código escrito mais genérico e reutilizável, economizando trabalho posterior.

Utilizando procedimentos e funções definidas pelo usuário

Os procedimentos e as funções permitem que você guarde o código utilizado com muita freqüência em um único local e possa chamá-lo em qualquer parte do aplicativo quando precisar. Isso facilita a leitura e a manutenção do seu código, pois uma alteração pode ser feita uma única vez no procedimento, não sendo necessário efetuá-la várias vezes em seus programas.

No Visual FoxPro, os procedimentos são deste tipo:

```
PROCEDURE myproc
  * Isto é um comentário, mas poderia ser um código executável
ENDPROC
```

Tradicionalmente, os procedimentos contêm códigos que você escreve para executar uma operação, e as funções realizam algumas operações e retornam um valor. No Visual FoxPro, as funções se assemelham aos procedimentos:

```
FUNCTION myfunc
  * Isto é um comentário mas poderia ser um código executável
ENDFUNC
```

Você pode incluir procedimentos e funções em um arquivo de programa separado ou no final de um arquivo de programa que contenha o código de programa normal. Você não pode ter um código de programa executável normal incluído em um arquivo de programa após procedimentos e funções.

Se incluir seus procedimentos e funções em um arquivo de programa separado, você poderá torná-los acessíveis ao seu programa utilizando o comando SET PROCEDURE TO. Por exemplo, se você tiver um arquivo denominado FUNPROC.PRG, utilize o comando a seguir na janela **Comando**:

```
SET PROCEDURE TO funproc.prg
```

Chamando um procedimento ou função

Há duas formas de chamar um procedimento ou uma função em seus programas:

- Utilize o comando DO. Por exemplo:

  ```
  DO myproc
  ```

  – Ou –

- Inclua um conjunto de parênteses depois do nome da função. Por exemplo:

  ```
  myfunc( )
  ```

Cada um desses métodos pode ser expandido enviando ou recebendo valores do procedimento ou da função.

Enviando valores a um procedimento ou função

Para enviar valores a procedimentos ou funções, você pode incluir parâmetros. Por exemplo, o procedimento a seguir aceita um único parâmetro:

```
PROCEDURE myproc( cString )
   * A linha a seguir exibe uma mensagem
   MESSAGEBOX ("myproc" + cString)
ENDPROC
```

Observação A inclusão dos parâmetros entre parênteses em uma linha de definição de procedimento ou função, por exemplo, `PROCEDURE myproc(cString)`, indica que o parâmetro tem um *escopo* local em relação ao procedimento ou função. Você também pode permitir que uma função ou um procedimento aceite parâmetros de escopo local utilizando LPARAMETERS.

Os parâmetros funcionam da mesma forma em uma função. Para enviar um valor como um parâmetro a este procedimento ou a uma função, você pode utilizar uma seqüência ou uma variável que contenha uma seqüência, conforme indicado na tabela a seguir.

Passagem de parâmetros

Código	Comentários
`DO myproc WITH cTestString` `DO myproc WITH " seqüência de teste"`	Chama um procedimento e passa uma seqüência literal ou uma variável de caracteres.
`Myfunc("seqüência  de teste")` `myfunc( cTestString )`	Chama uma função e passa uma cópia de uma variável de caracteres ou seqüência literal.

Observação Se você chamar um procedimento ou uma função sem utilizar o comando DO, a definição UDFPARMS controlará como os parâmetros são passados. Como padrão, UDFPARMS é definido como VALUE para que cópias dos parâmetros sejam passadas. Quando você utilizar DO, o parâmetro real é utilizado (o parâmetro é passado por referência) e quaisquer alterações no procedimento ou na função são refletidas nos dados originais, independentemente da definição de UDFPARMS.

Você pode enviar diversos valores a um procedimento ou a uma função, separando-os por vírgulas. Por exemplo, o procedimento a seguir pressupõe três parâmetros: uma data, uma seqüência de caracteres e um número.

```
PROCEDURE myproc( dDate, cString, nTimesToPrint )
   FOR nCnt = 1 to nTimesToPrint
      ? DTOC(dDate) + " " + cString + " " + STR(nCnt)
   ENDFOR
ENDPROC
```

Você poderia chamar este procedimento com a linha de código a seguir:

```
DO myproc WITH DATE(), "Olá Mundo", 10
```

Recebendo valores de uma função

O valor de retorno padrão é verdadeiro (.T.) mas você pode utilizar o comando RETURN para retornar qualquer valor. Por exemplo, a função a seguir retorna uma data duas semanas após a data passada para ela como um parâmetro.

```
FUNCTION plus2weeks
PARAMETERS dDate
   RETURN dDate + 14
ENDFUNC
```
A linha de código a seguir armazena o valor retornado por essa função em uma variável:

```
dDeadLine = plus2weeks(DATE())
```
A tabela abaixo lista as formas de armazenar ou exibir valores retornados por uma função:

Manipulando valores de retorno

Código	Comentários
`var = myfunc( )`	Armazena o valor retornado pela função em uma variável.
`? myfunc( )`	Imprime o valor retornado pela função na janela de saída ativa.

Verificando parâmetros em um procedimento ou função

É uma boa idéia verificar se os parâmetros enviados ao seu procedimento ou função são os que você espera receber. Você pode utilizar as funções TYPE( ) e PARAMETERS( ) para verificar o tipo e o número de parâmetros enviados ao procedimento ou à função.

O exemplo na seção anterior, por exemplo, precisa receber um parâmetro do tipo Data. Você pode utilizar a função TYPE( ) para certificar-se de que o valor recebido pela função é do tipo certo.

```
FUNCTION plus2weeks( dDate )
   IF TYPE("dDate") = "D"
      RETURN dDate + 14
   ELSE
      MESSAGEBOX( "Você deve passar uma data!" )
      RETURN {}      && Retorna uma data vazia
   ENDIF
ENDFUNC
```
Se um procedimento receber mais parâmetros do que o esperado, o Visual FoxPro irá gerar uma mensagem de erro. Por exemplo, se tiver listado dois parâmetros e chamar o procedimento com três parâmetros, você receberá uma mensagem de erro. Contudo, se um procedimento receber menos parâmetros do que o esperado, os parâmetros adicionais são simplesmente inicializados com um valor falso (.F.). Como não há uma forma de saber se o último parâmetro foi definido como falso (.F.) ou omitido, o procedimento a seguir irá verificar se o número apropriado de parâmetros foi enviado:

```
PROCEDURE SaveValue( cStoreTo, cNewVal, lIsInTable )
   IF PARAMETERS( ) < 3
      MESSAGEBOX( "Número insuficiente de parâmetros passado." )
      RETURN .F.
   ENDIF
   IF lIsInTable
      REPLACE (cStoreTo) WITH (cNewVal)
   ELSE
      &cStoreTo = cNewVal
   ENDIF
   RETURN .T.
ENDPROC
```

Convertendo o programa NUMONLY em uma função

O programa de exemplo NUMONLY.PRG discutido na seção O processo de programação, pode se tornar mais eficaz e útil através da criação de uma função para a parte do programa que remove os caracteres não-numéricos de uma seqüência.

Exemplo de procedimento para retornar caracteres numéricos de uma seqüência

Código	Comentários
`FUNCTION NumbersOnly( cMixedVal )`	Início da função, que aceita uma seqüência de caracteres.

`CNumOnly = ""`
`FOR nCnt = 1 TO LEN(cMixedVal)`
`cCharacter = ;`
`SUBSTR(cMixedVal, nCnt, 1)`
`IF ISDIGIT(cCharacter)`
`cNumOnly = ;`
`cNumOnly + cCharacter`
`ENDIF`
`ENDFOR`
	Cria uma seqüência que só contém os caracteres numéricos da seqüência original.

`RETURN cNumOnly`	Retorna a seqüência que só contém caracteres numéricos.
`ENDFUNC`	Final da função.

Além de permitir a utilização deste código em várias situações, esta função facilita a leitura do programa:

```
SCAN
   REPLACE FieldName WITH NumbersOnly(FieldName)
ENDSCAN
```

Ou mais simples ainda:

```
REPLACE ALL FieldName WITH NumbersOnly(FieldName)
```

Seguindo adiante

A programação procedural, juntamente com a programação orientada a objetos e com as ferramentas de criação do Visual FoxPro, podem ajudá-lo a desenvolver um aplicativo versátil do Visual FoxPro. O restante deste manual traz tópicos com os quais você irá se deparar ao desenvolver aplicativos do Visual FoxPro.

Para obter maiores informações sobre a programação com uma abordagem orientada a objetos, consulte o capítulo 3, Programação orientada a objetos. Para aprender mais sobre a criação de formulários com o **Criador de formulários**, consulte o capítulo 9, Criando formulários.