# Técnicas de Programação

***TP1001***

**Prof. Giovane Barcelos**
giovane_barcelos@uniritter.edu.br

# Plano de Ensino
## Conteúdo programático

| | |
|---|---|
| 1. Introdução à programação em C<br>2. Desenvolvimento estruturado de programas em C<br>3. Controle de programa<br>4. Funções<br>5. Arrays<br>6. Ponteiros | **N1** |
| 7. Caracteres e strings<br>8. Entrada/Saída formatada<br>9. Estruturas, uniões, manipulações de bits e enumerações<br>10. Processamento de arquivos<br>11. Estruturas de dados<br>12. O pré-processador<br>13. Outros tópicos sobre C | **N2** |

# Objetivos

- A criar, ler, gravar e atualizar arquivos.
- Processar arquivos por acesso sequencial.
- Processar arquivos por acesso aleatório.

# Introdução

- O armazenamento de dados em variáveis e arrays é temporário — esses dados são perdidos quando um programa é encerrado.
- **Arquivos** são usados na conservação permanente de dados.
- Os computadores armazenam arquivos em dispositivos secundários de armazenamento, especialmente dispositivos de armazenamento de disco.
- Neste capítulo, explicaremos como os arquivos de dados são criados, atualizados e processados por programas em C.
- Examinaremos tanto os arquivos de acesso sequencial quanto os arquivos de acesso aleatório.

# Hierarquia de dados

- Basicamente, todos os dados processados por um computador são reduzidos a combinações de zeros e uns.
- Isso ocorre porque é simples e econômico construir dispositivos eletrônicos que podem assumir dois estados permanentes — um deles representa 0, e o outro representa 1.
- É notável que as impressionantes funções executadas pelos computadores envolvam apenas as manipulações mais elementares de 0s e 1s.
- O menor item de dados em um computador pode assumir o valor 0 ou o valor 1.

# Hierarquia de dados

- Tal item de dados é chamado de bit (abreviação de 'binary digit', ou 'dígito binário' — um dígito que pode assumir um de dois valores).
- Os circuitos computacionais realizam várias manipulações simples de bits, tais como determinar, redefinir e inverter o valor de um bit (de 1 para 0 ou de 0 para 1).
- It's cumbersome to work with data in the low-level form of bits.
- É complicado trabalhar com dados no formato de bits.
- Em vez disso, os programadores preferem trabalhar com dados na forma de dígitos decimais (ou seja, 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9), letras (ou seja, de A até Z, e de a até z) e símbolos especiais (ou seja, $, @, %, &, *, (, ), –, +,", :, ?, /, entre outros).

# Hierarquia de dados

- Dígitos, letras e símbolos especiais são chamados de caracteres.
- O conjunto de todos os caracteres usados para escrever programas e representar itens de dados em determinado computador é chamado de conjunto de caracteres daquele computador.
- Como os computadores podem processar apenas 1s e 0s, qualquer caractere do conjunto de caracteres de um computador é representado por uma combinação de 1s e 0s (chamada de byte).
- Atualmente, os bytes são normalmente compostos por oito bits. Os programadores criam programas e itens de dados como caracteres; os computadores manipulam e processam esses caracteres como combinações de bits..

# Hierarquia de dados

- Da mesma forma que os caracteres são compostos por bits, os campos são compostos por caracteres.
- Um campo é um grupo de caracteres que possui um significado.
- Por exemplo, um campo que tenha apenas letras maiúsculas e minúsculas pode ser usado para representar o nome de uma pessoa.
- Os itens de dados processados pelos computadores formam uma hierarquia de dados, na qual os itens de dados se tornam maiores e mais complexos na estrutura à medida que evoluímos de bits para caracteres (bytes), campos, e assim por diante.
- Um registro (isto é, uma struct em C) é composto por vários campos.

# Hierarquia de dados

- **Em um sistema de folha de pagamento, por exemplo, um registro de determinado funcionário pode consistir nos seguintes campos:**
  - **Número de identificação (campo alfanumérico).**
  - **Nome (campo alfabético).**
  - **Endereço (campo alfanumérico).**
  - **Valor do salário por hora (campo numérico).**
  - **Número de dispensas solicitadas (campo numérico).**
  - **Total de vencimentos no ano (campo numérico).**
  - **Total de impostos retidos na fonte (campo numérico).**

# Hierarquia de dados

- Assim, um registro é um grupo de campos relacionados.
- No exemplo anterior, cada um dos campos pertence ao mesmo funcionário.
- Naturalmente, uma empresa específica pode ter muitos funcionários, e, portanto, terá um registro de folha de pagamento para cada um.
- Um arquivo é um grupo de registros relacionados

# Hierarquia de dados

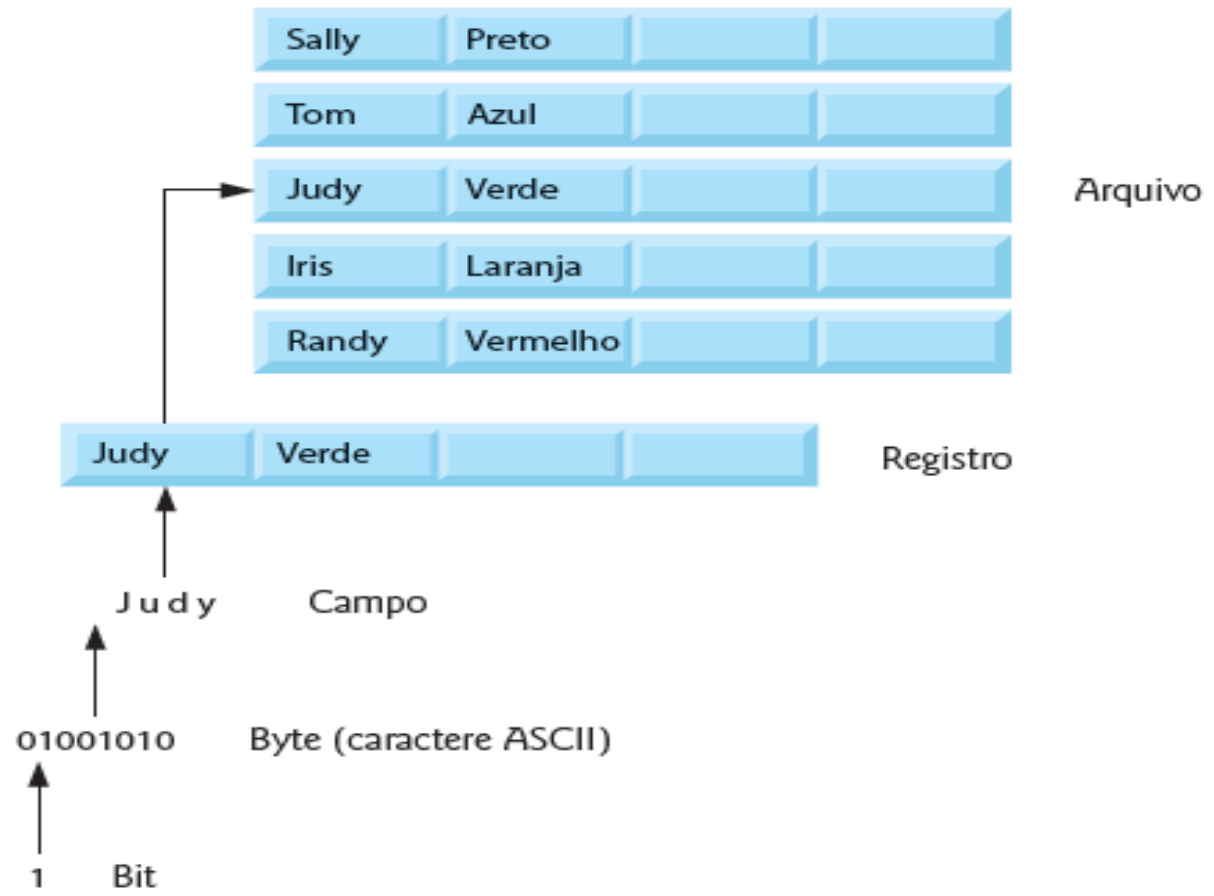


Figura 11.1 ■ A hierarquia de dados.

# Hierarquia de dados

- Um arquivo de folha de pagamento de uma empresa contém um registro para cada funcionário.
- Assim, o arquivo de folha de pagamento de uma pequena empresa pode conter apenas 22 registros, ao passo que o arquivo da folha de pagamento de uma grande empresa pode conter 100.000 registros.
- Não é raro uma organização ter centenas ou até milhares de arquivos, alguns deles contendo bilhões ou trilhões de caracteres de informações.
- A Figura 11.1 ilustra a hierarquia de dados.

# Hierarquia de dados

- Para facilitar a recuperação de registros específicos de um arquivo, pelo menos um campo em cada registro é escolhido para ser a chave de registro.
- A chave de registro identifica e relaciona um registro a determinada pessoa ou entidade.
- Por exemplo, no registro de folha de pagamento descrito anteriormente, o número de identificação do empregado normalmente seria escolhido para ser a chave de registros.
- Há muitas maneiras de se organizar registros em um arquivo.
- No tipo mais comum de organização, denominado arquivo sequencial, registros são normalmente organizados segundo o campo de chave de registros.

# Hierarquia de dados

- Em um arquivo de folha de pagamento, os registros geralmente são organizados segundo o número de identificação do funcionário.
- O registro do primeiro empregado no arquivo contém o menor número de identificação, e os registros subsequentes têm números de identificação em ordem crescente.
- As empresas podem ter arquivos de folha de pagamento, de contas a receber (que lista o dinheiro devido pelos clientes), de contas a pagar (que lista o dinheiro devido aos fornecedores), de estoques (que lista as características a respeito de todos os itens manipulados pela empresa), e muitos outros tipos de arquivos..
- Algumas vezes, um grupo de arquivos relacionados entre si é chamado de **banco de dados**.
- Um conjunto de programas que se destina a criar e gerenciar bancos de dados é chamado de **SGBD, sistema de gerenciamento de banco de dados** (ou DBMS, database management system).

# Arquivos e streams

- C vê cada arquivo simplesmente como uma sequência de bytes (Figura 11.2).
- Cada arquivo termina com um marcador de fim de arquivo, ou em um byte específico, cujo número é gravado em uma estrutura administrativa de dados mantida pelo sistema.
- Quando um arquivo é aberto, um objeto é criado e um stream é associado àquele objeto.
- Três arquivos e seus streams associados são abertos automaticamente quando um programa inicia sua execução — a entrada-padrão, a saída-padrão e o erro-padrão.
- Os streams fornecem canais de comunicação entre arquivos e programas

# Arquivos e streams

- Por exemplo, o stream de entrada-padrão permite que um programa leia dados do teclado, e o fluxo de saída-padrão permite que um programa exiba dados na tela.
- Abrir um arquivo retorna um ponteiro para uma estrutura FILE (definida em <stdio.h>) , que contém informações usadas para processar o arquivo.
- Essa estrutura inclui um descritor de arquivo, ou seja, um índice para um array do sistema operacional chamado de tabela de arquivo aberto.
- Cada elemento do array contém um bloco de controle de arquivo (File Control Block — FCB), que o sistema operacional utiliza para administrar um arquivo específico.
- Entrada-padrão, saída-padrão e erro-padrão são manipulados usando os ponteiros de arquivo stdin, stdout e stderr.

# Arquivos e streams


Figura 11.2 ■ Como a linguagem C visualiza um arquivo com *n* bytes.

# Arquivos e streams

- A biblioteca-padrão oferece muitas funções para a leitura de dados dos arquivos e para a gravação de dados em arquivos.
- A função fgetc, assim como getchar, lê um caractere de um arquivo.
- A função fgetc recebe como argumento um ponteiro FILE para o arquivo, do qual um caractere será lido.
- A chamada fgetc( stdin ) lê um caractere de stdin — a entrada-padrão.
- Essa chamada é equivalente à chamada getchar().
- A função fputc, assim como putchar, grava um caractere em um arquivo.
- A função fputc recebe como argumentos um caractere a ser gravado e um ponteiro para o arquivo no qual o caractere será gravado.

# Arquivos e streams

- A chamada de função fputc( 'a', stdout ) envia o caractere 'a' para stdout — a saída-padrão.
- Essa chamada é equivalente putchar( 'a' ).
- Várias outras funções usadas na leitura de dados da entrada-padrão e no envio de dados para a saída-padrão possuem funções de processamento de arquivo com nomes semelhantes.
- As funções fgets e fputs, por exemplo, podem ser usadas na leitura de uma linha de um arquivo e gravar uma linha em um arquivo, respectivamente.
- Nas próximas seções, apresentaremos os equivalentes de processamento de arquivo das funções scanf e printf — fscanf e fprintf.

# Criação de um arquivo de acesso sequencial

- C não impõe nenhuma estrutura a um arquivo.
- Assim, conceitos como o de ‘registro’ não fazem parte da linguagem em C.
- Portanto, o programador deve estruturar os arquivos de modo a satisfazer as exigências de uma aplicação em particular.
- No exemplo a seguir,  vemos como o programador pode impor uma estrutura de registros simples a um arquivo.
- A Figura 11.3 cria um arquivo simples de acesso sequencial que pode ser usado em um sistema de contas a receber para ajudar a controlar as quantias devidas pelos clientes devedores de uma empresa.

# Criação de um arquivo de acesso sequencial

- **Para cada cliente, o programa obtém um número de conta, o nome e o saldo do cliente (ou seja, a quantia que ele deve à empresa por bens e serviços recebidos no passado).**
- **Os dados obtidos constituem um 'registro' daquele cliente.**
- **O número da conta é usado como campo de chave dos registros nessa aplicação — o arquivo será criado e mantido segundo a ordem dos números de contas.**

# Criação de um arquivo de acesso sequencial

```
1  /* Fig. 11.3: fig11_03.c
2     Criando um arquivo sequencial */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int account; /* número da conta */
8     char name[ 30 ]; /* nome da conta */
9     double balance; /* saldo da conta */
10
11    FILE *cfPtr; /* ponteiro de arquivo cfPtr = clients.dat */
12
13    /* fopen abre arquivo. Sai do programa se não pode criar arquivo  */
14    if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15       printf( "Arquivo não pode ser aberto\n" );
16    } /* fim do if */
```

Figura 11.3 ■ Criação de um arquivo sequencial. (Parte 1 de 2.)

# Criação de um arquivo de acesso sequencial

```
17      else {
18         printf( "Digite o número de conta, o nome e o saldo.\n" );
19         printf( "Digite fim de arquivo para terminar a entrada.\n" );
20         printf( "? " );
21         scanf( "%d%s%lf", &account, name, &balance );
22
23      /* grava conta, nome e saldo no arquivo com fprintf */
24      while ( !feof( stdin ) ) {
25         fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26         printf( "? " );
27         scanf( "%d%s%lf", &account, name, &balance );
28      } /* fim do while */
29
30         fclose( cfPtr ); /* fclose fecha arquivo */
31      } /* fim do else */
32
33      return 0; /* indica conclusão bem-sucedida */
34   } /* fim do main */
```

```
Digite o número de conta, o nome e o saldo.
Digite fim de arquivo para terminar a entrada.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Figura 11.3 ■ Criação de um arquivo sequencial. (Parte 2 de 2.)

# Criação de um arquivo de acesso sequencial

- **Esse programa supõe que o usuário forneça os registros na ordem dos números das contas.**
- **Em um grande sistema de contas a receber, seria fornecido um recurso de ordenação para que o usuário pudesse entrar com os registros em qualquer ordem.**
- **Os registros seriam, então, ordenados e gravados no arquivo.**
- **[*Nota*: as figuras 11.7 e 11.8 usam o arquivo de dados criado na Figura 11.3, assim, você precisa executar o programa da Figura 11.3 antes dos programas nas figuras 11.7 e 11.8.]**

# Criação de um arquivo de acesso sequencial

- Agora, examinemos esse programa.
- A linha 11 indica que cfptr é um ponteiro para uma estrutura FILE.
- Um programa em C administra cada arquivo com uma estrutura FILE separada.
- Você não precisa conhecer os detalhes da estrutura FILE para usar arquivos, embora o leitor interessado possa estudar a declaração em stdio.h.
- Logo, veremos exatamente como a estrutura FILE segue indiretamente na direção do bloco de controle de arquivo (FCB) do sistema operacional de um arquivo.
- Cada arquivo aberto precisa ter um ponteiro do tipo FILE declarado separadamente, que é usado para se referir ao arquivo.

# Criação de um arquivo de acesso sequencial

- A linha 14 dá nome ao arquivo — "clients.dat" — a ser usado pelo programa para estabelecer uma 'linha de comunicação' com o arquivo.
- O ponteiro de arquivo cfPtr recebe um ponteiro para a estrutura FILE para o arquivo aberto com fopen.
- A função fopen usa dois argumentos: um nome de arquivo e um modo de abertura de arquivo.
- O modo de abertura de arquivo "w" indica que o arquivo deve ser aberto para gravação (ou escrita).
- Se um arquivo não existir e ele for aberto para gravação, fopen criará o arquivo.

# Criação de um arquivo de acesso sequencial

- Se um arquivo existente for aberto para gravação, o conteúdo do arquivo será descartado sem aviso.
- No programa, a estrutura if é usada para determinar se o ponteiro de arquivo cfPtr é **NULL** (ou seja, o arquivo não está aberto).
- Se ele for NULL, o programa exibirá uma mensagem de erro e será encerrado.
- Caso contrário, o programa processará a entrada e a gravará no arquivo.

# Criação de um arquivo de acesso sequencial

**Erro comum de programação 11.1**
*Abrir um arquivo existente para gravação (“w”) quando, na verdade, o usuário deseja preservar o arquivo, descarta o conteúdo do arquivo sem aviso.*

**Erro comum de programação 11.2**
*Esquecer de abrir um arquivo antes de tentar referenciá-lo em um programa é um erro lógico.*

# Criação de um arquivo de acesso sequencial

- O programa pede que o usuário digite os diversos campos para cada registro ou digite o fim de arquivo quando a entrada de dados for completada.
- A Figura 11.4 lista as principais combinações de teclas para a entrada do fim de arquivo em diversos sistemas de computação.
- A linha 24 usa a função feof para determinar se o indicador de fim de arquivo está definido para o arquivo ao qual o stdin se refere.
- O indicador de fim de arquivo informa ao programa que não existem mais dados a serem processados.
- Na Figura 11.3, um indicador de fim de arquivo é definido como entrada-padrão quando o usuário entra com a combinação de teclas de fim de arquivo.
- O argumento para a função feof ié um ponteiro para o arquivo cujo indicador de fim de arquivo está sendo testado (nesse caso, stdin).

# Criação de um arquivo de acesso sequencial

| Sistema operacional | Combinação de teclas |
|---|---|
| Linux/Mac OS X/UNIX | *<Ctrl> d* |
| Windows | *<Ctrl> z* |

Figura 11.4 ■ Combinações de teclas para fim de arquivo usadas em diversos sistemas operacionais populares.

# Criação de um arquivo de acesso sequencial

- A função retornará um valor diferente de zero (verdadeiro) quando o indicador de fim de arquivo for definido; caso contrário, a função retornará zero.
- A estrutura while, que inclui a chamada a feof nesse programa, continuará a ser executada enquanto o indicador de fim de arquivo não for definido.
- A linha 25 grava os dados no arquivo clients.dat.
- Os dados podem ser recuperados mais tarde por um programa preparado para ler o arquivo (ver Seção 11.5).

# Criação de um arquivo de acesso sequencial

- A função fprintf é equivalente a printf, mas fprintf também recebe como argumento um ponteiro de arquivo para o arquivo no qual os dados serão gravados.
- A função fprintf pode enviar dados para a saída-padrão usando stdout como ponteiro de arquivo, como em:
  - fprintf( stdout, "%d %s %.2f\n",
    account, name, balance );

# Criação de um arquivo de acesso sequencial

**Erro comum de programação 11.3**
Usar o ponteiro de arquivo errado para se referir a um arquivo é um erro lógico.

**Dica de prevenção de erro 11.1**
*Certifique-se de que, em um programa, as chamadas para as funções de processamento de arquivo contenham os*
*ponteiros de arquivo corretos.*

# Criação de um arquivo de acesso sequencial

- Depois que o usuário digita o indicador de fim de arquivo, o programa fecha o arquivo clients.dat com fclose e termina.
- A função fclose também recebe o ponteiro de arquivo (em vez do nome de arquivo) como um argumento.
- Se a função fclose não for chamada explicitamente, o sistema operacional normalmente fechará o arquivo quando a execução do programa finalizar.
- Este é um exemplo de 'manutenção' do sistema operacional.

# Criação de um arquivo de acesso sequencial

**Boa prática de programação 11.1**
*Feche, explicitamente, cada arquivo assim que ele não for mais necessário.*

**Dica de desempenho 11.1**
*Fechar um arquivo pode liberar recursos pelos quais outros usuários ou programas podem estar esperando.*

# Criação de um arquivo de acesso sequencial

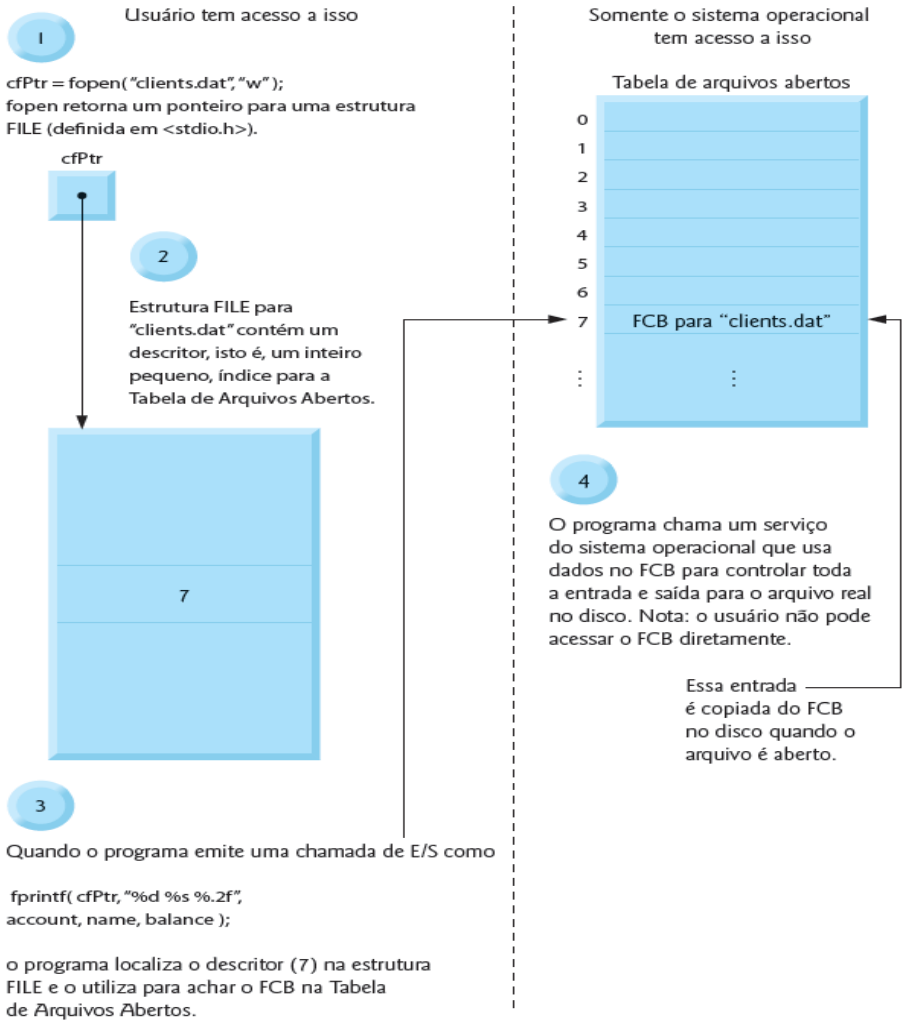


Figura 11.5 ■ Relação entre ponteiros FILE, estruturas FILE e FCBs.

# Criação de um arquivo de acesso sequencial

- Na execução do exemplo no programa da Figura 11.3, o usuário digita informações para cinco contas, depois digita o indicador de fim de arquivo para sinalizar que a entrada de dados foi completada.
- O exemplo não mostra como os registros de dados realmente aparecem no arquivo.
- Para verificar se o arquivo foi criado com sucesso, apresentaremos, na próxima seção, um programa que lê o arquivo e exibe seu conteúdo.
- A Figura 11.5 ilustra a relação entre os ponteiros FILE, estruturas FILE e FCBs (bloco de controle de arquivo) na memória.
- Quando o arquivo "clients.dat" é aberto, um FCB para o arquivo é copiado para a memória.

# Criação de um arquivo de acesso sequencial

- A figura mostra a conexão entre o ponteiro de arquivo retornado por fopen e o FCB usado pelo sistema operacional para administrar o arquivo.
- Os programas podem não processar nenhum arquivo, um arquivo ou vários arquivos.
- Cada arquivo usado em um programa precisa ter um nome exclusivo, e terá um ponteiro de arquivo diferente retornado por fopen.
- Todas as funções de processamento de arquivos subsequentes após o arquivo ter sido aberto precisam se referir ao arquivo com o ponteiro de arquivo apropriado.
- Os arquivos podem ser abertos de vários modos (Figura 11.6).
- Para criar um arquivo, ou para descartar o conteúdo de um arquivo antes de gravar dados, abra o arquivo para gravação("w").

# Criação de um arquivo de acesso sequencial

- Para ler um arquivo existente, abra-o para leitura ("r").
- Para acrescentar registros ao final de um arquivo existente, abra o arquivo para acréscimo ("a").
- Para abrir um arquivo de modo que possa ser gravado e lido, abra o arquivo para atualização em um dos três modos de atualização — "r+", "w+" ou "a+".
- O modo "r+" abre um arquivo para leitura e gravação.
- O modo "w+" cria um arquivo para leitura e gravação.
- Se o arquivo já existir, ele é aberto e seu conteúdo, descartado.

# Criação de um arquivo de acesso sequencial

| Modo | Descrição |
|---|---|
| r | Abre um arquivo existente para leitura. |
| w | Cria um arquivo para gravação. Se o arquivo já existe, descarta o conteúdo atual. |
| a | Acréscimo; abre ou cria um arquivo para gravação no final do arquivo. |
| r+ | Abre um arquivo existente para atualização (leitura e gravação). |
| w+ | Cria um arquivo para atualização. Se o arquivo já existe, descarta o conteúdo atual. |
| a+ | Acréscimo: abre ou cria um arquivo para atualização; a gravação é feita no final do arquivo. |
| rb | Abre um arquivo existente para leitura no modo binário. |
| wb | Cria um arquivo para gravação no modo binário. Se o arquivo já existe, descarta o conteúdo atual. |
| ab | Acréscimo; abre ou cria um arquivo para gravação no final do arquivo no modo binário. |
| rb+ | Abre um arquivo existente para atualização (leitura e gravação) no modo binário. |
| wb+ | Cria um arquivo para atualização no modo binário. Se o arquivo já existir, descarta o conteúdo atual. |
| ab+ | Acréscimo: abre ou cria um arquivo para atualização no modo binário; a gravação é feita no final do arquivo. |

Figura 11.6 ■ Modos de abertura de arquivo.

# Criação de um arquivo de acesso sequencial

- O modo "a+" abre um arquivo para leitura e gravação — toda gravação é feita ao final do arquivo.
- Se o arquivo não existe, ele é criado.
- Cada modo de abertura de arquivo possui um modo binário correspondente (que contém a letra b), que manipula arquivos binários.
- Os modos binários serão usados nas seções 11.6 a 11.10, quando introduziremos os arquivos de acesso aleatório.
- Se houver um erro enquanto um arquivo estiver sendo aberto em um desses modos, fopen retorna NULL.

# Criação de um arquivo de acesso sequencial

**Erro comum de programação 11.4**
*Abrir um arquivo inexistente para leitura.*

**Erro comum de programação 11.5**
*Abrir um arquivo para leitura ou gravação sem ter recebido os direitos de acesso apropriados para esse arquivo (isso depende do sistema operacional).*

**Erro comum de programação 11.6**
*Abrir um arquivo para gravação quando não existe espaço disponível no disco.*

# Criação de um arquivo de acesso sequencial

**Erro comum de programação 11.7**

*Abrir um arquivo com o modo de arquivo incorreto é um erro lógico. Por exemplo, abrir um arquivo no modo de gravação (*"w"*) quando ele deveria ser aberto no modo de atualização (*"r+"*) faz com que o conteúdo anterior do arquivo seja descartado.*

**Dica de prevenção de erro 11.2**

*Abra um arquivo somente para leitura (e não para atualização) se o conteúdo do arquivo não tiver de ser modificado. Isso evita a modificação inadvertida do conteúdo do arquivo. Esse é outro exemplo do princípio do menor privilégio.*

# Leitura de dados de um arquivo de acesso sequencial

```
1  /* Fig. 11.7: fig11_07.c
2     Lendo e imprimindo um arquivo sequencial */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int account; /* número da conta */
8     char name[ 30 ]; /* nome da conta */
9     double balance;  /* saldo da conta */
10
11    FILE *cfPtr;   /* ponteiro de arquivo cfPtr = clients.dat */
12
13    /* fopen abre arquivo; sai do programa se o arquivo não puder ser aberto */
14    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
15       printf( "arquivo não pode ser aberto\n" );
16    } /* fim do if */
17    else { /* lê conta, nome e saldo do arquivo*/
18       printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
19       fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
20
```

# Leitura de dados de um arquivo de acesso sequencial

```
21      /* enquanto não é fim de arquivo */
22      while ( !feof( cfPtr ) ) {
23      printf( "%-10d%-13s%7.2f\n", account, name, balance );
24      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
25      } /* fim do while */
26
27      fclose( cfPtr ); /* fclose fecha o arquivo */
28      } /* fim do else */
29
30      return 0; /* indica conclusão bem-sucedida */
31  } /* fim do main */
```

```
Conta    Nome      Saldo
100      Jones       24.98
200      Doe        345.67
300      White        0.00
400      Stone      -42.16
500      Rich       224.62
```

Figura 11.7 ■ Leitura e impressão de um arquivo sequencial.

# Criação de um arquivo de acesso sequencial

- Os dados são armazenados em arquivos de modo que possam ser recuperados para processamento quando necessário.
- A seção anterior demonstrou como criar um arquivo de acesso sequencial.
- Nesta seção, analisaremos como ler dados de um arquivo sequencialmente.
- O programa da Figura 11.7 lê registros do arquivo "clients.dat" criados pelo programa da Figura 11.3, e exibe o conteúdo desses registros.
- A linha 11 indica que cfPtr é um ponteiro para um FILE.
- A linha 14 tenta abrir o arquivo "clients.dat" para leitura ("r"), e determina se o arquivo foi aberto com sucesso (ou seja, fopen não retorna NULL).

# Leitura de dados de um arquivo de acesso sequencial

- A linha 19 lê um ‘registro’ do arquivo.
- A função fscanf é equivalente à função scanf, exceto que fscanf recebe como argumento um ponteiro de arquivo para o arquivo do qual os dados foram lidos
- Depois que essa instrução for executada pela primeira vez, account terá o valor 100, name terá o valor "Jones" e balance terá o valor 24.98.
- Toda vez que a segunda instrução fscanf (linha 24) for executada, o programa lerá outro registro do arquivo, e account, name e balance assumirão novos valores.
- Quando o programa atinge o final do arquivo, o arquivo é fechado (linha 27), e o programa termina.
- A função feof retorna verdadeira somente *depois* que o programa tenta ler os dados inexistentes após a última linha.

# Leitura de dados de um arquivo de acesso sequencial

- **Para recuperar dados de um arquivo sequencialmente, os programas em geral começam a ler a partir do início do arquivo, e leem todos os dados, um após o outro, até que os dados desejados sejam encontrados.**
- **Pode ser necessário processar o arquivo sequencialmente várias vezes (desde o seu início) durante a execução de um programa.**

# Leitura de dados de um arquivo de acesso sequencial

- Uma instrução como
  - rewind( cfPtr );

  faz com que o ponteiro de posição do arquivo — que indica o número do próximo byte a ser lido ou gravado no arquivo — seja reposicionado para o início do arquivo (ou seja, para o byte 0) apontado por cfPtr.
- O ponteiro de posição do arquivo não é realmente um ponteiro.
- Na verdade, ele é um valor inteiro que especifica o local do byte no arquivo em que ocorrerá a próxima leitura ou gravação.
- Isso, às vezes, é chamado de deslocamento de arquivo.
- O ponteiro de posição do arquivo é um membro da estrutura FILE associado a cada um dos arquivos.

# Leitura de dados de um arquivo de acesso sequencial

```
1  /* Fig. 11.8: fig11_08.c
2     Programa de consulta de crédito */
3  #include <stdio.h>
4
5  /* função main inicia execução do programa */
6  int main( void )
7  {
8     int request; /* número de solicitação */
9     int account; /* número de conta */
10    double balance;  /* saldo da conta */
11    char name[ 30 ]; /* nome da conta */
12    FILE *cfPtr; /* ponteiro de arquivo clients.dat */
13
14    /* fopen abre o arquivo; sai do programa se arquivo não abre */
15    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
16       printf( "Arquivo não pode ser aberto\n" );
17    } /* fim do if */
18    else {
19
20       /* exibe opções de requerimento */
21       printf( "Digite solicitação\n"
22          " 1 - Lista contas com saldo zero\n"
23          " 2 - Lista contas com saldo credor\n"
24          " 3 - Lista contas com saldo devedor\n"
25          " 4 - Fim da execução\n? " );
26       scanf( "%d", &request );
27
```

# Leitura de dados de um arquivo de acesso sequencial

```
28          /* processa solicitação do usuário */
29          while ( request != 4 ) {
30
31             /* lê conta, nome e saldo do arquivo */
32             fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
33
34             switch ( request ) {
35                case 1:
36                   printf( "\nContas com saldo zero:\n" );
37
38                   /* lê conteúdo do arquivo (até eof) */
39                   while ( !feof( cfPtr ) ) {
```

Figura 11.8 ■ Programa de consulta de crédito. (Parte 1 de 2.)

# Leitura de dados de um arquivo de acesso sequencial

```
40
41                   if ( balance == 0 ) {
42                      printf( "%-10d%-13s%7.2f\n",
43                         account, name, balance );
44                   } /* fim do if */
45
46                   /* lê conta, nome e saldo do arquivo */
47                   fscanf( cfPtr, "%d%s%lf",
48                      &account, name, &balance );
49                } /* fim do while */
50
51                break;
52             case 2:
53                printf( "\nContas com saldo credor:\n" );
54
55                /* lê conteúdo do arquivo (até eof) */
56                while ( !feof( cfPtr ) ) {
57
58                   if ( balance < 0 ) {
59                      printf( "%-10d%-13s%7.2f\n",
60                         account, name, balance );
61                   } /* fim do if */
62
63                   /* lê conta, nome e saldo do arquivo */
64                   fscanf( cfPtr, "%d%s%lf",
65                      &account, name, &balance );
66                } /* fim do while */
67
68                break;
69             case 3:
70                printf( "\nContas com saldo devedor:\n" );
71
72                /* lê conteúdo do arquivo (até eof) */
73                while ( !feof( cfPtr ) ) {
74
```

# Leitura de dados de um arquivo de acesso sequencial

```
75                      if ( balance > 0 ) {
76                         printf( "%-10d%-13s%7.2f\n",
77                            account, name, balance );
78                      } /* fim do if */
79
80                      /* lê conta, nome e saldo do arquivo */
81                      fscanf( cfPtr, "%d%s%lf",
82                         &account, name, &balance );
83                   } /* fim do while */
84
85                   break;
86             } /* end switch */
87
88             rewind( cfPtr ); /* retorna cfPtr para início do arquivo */
89
90             printf( "\n? " );
91             scanf( "%d", &request );
92          } /* fim do while */
93
94          printf( "Fim da execução.\n" );
95          fclose( cfPtr ); /* fclose fecha o arquivo */
96       } /* fim do else */
97
98       return 0; /* indica conclusão bem-sucedida */
99    } /* fim do main */
```

Figura 11.8 ■ Programa de consulta de crédito. (Parte 2 de 2.)

# Leitura de dados de um arquivo de acesso sequencial

- O programa da Figura 11.8 permite que um gerente de crédito obtenha listas com nomes de clientes com saldo zero (que não devem dinheiro à empresa), clientes com saldo credor (para os quais a empresa deve dinheiro) e clientes com saldo devedor (que devem dinheiro à empresa por bens e serviços recebidos).
- Um saldo credor é um valor negativo; um saldo devedor é um valor positivo.

# Leitura de dados de um arquivo de acesso sequencial

```
Digite solicitação
 1 - Lista contas com saldo zero
 2 - Lista contas com saldo credor
 3 - Lista contas com saldo devedor
 4 - Fim da execução
? 1
Contas com saldo zero:
300       White                                0.00
? 2
Contas com saldo credor:
400       Stone                              -42.16
? 3
Contas com saldo devedor:
100       Jones                               24.98
200       Doe                                345.67
500       Rich                               224.62
? 4
Fim da execução.
```

Figura 11.9 ■ Exemplo de saída do programa de consulta de crédito da Figura 11.8.

# Leitura de dados de um arquivo de acesso sequencial

- O programa exibe um menu e permite ao gerente de crédito digitar uma de três opções para obter informações de crédito.
- A opção 1 produz uma lista de contas com saldo zero.
- A opção 2 produz uma lista de contas com saldo credor.
- A opção 3 produz uma lista de contas com saldo devedor.
- A opção 4 termina a execução do programa.
- A saída do programa é mostrada na Figura 11.9.

# Leitura de dados de um arquivo de acesso sequencial

- Os dados nesse tipo de arquivo sequencial não podem ser modificados sem que se corra o risco de destruir outros dados.
- Por exemplo, se o nome “White” precisa ser mudado para “Worthington”, o novo nome não pode ser, simplesmente, gravado em cima do antigo.
- O registro para White foi gravado no arquivo como

# Leitura de dados de um arquivo de acesso sequencial

- Se esse registro é regravado, começando na mesma posição no arquivo e usando o nome mais longo, o registro seria
  - 300 Worthington 0.00
- O novo registro é maior (tem mais caracteres) que o registro original.
- Os caracteres além do segundo “o” em “Worthington” seriam gravados sobre o início do próximo registro sequencial no arquivo.
- Nesse caso, o problema é que, no modelo de entrada/saída formatada usando fprintf e fscanf, os campos — e, portanto, os registros — podem variar em tamanho.

# Leitura de dados de um arquivo de acesso sequencial

- Por exemplo, os valores 7, 14, –117, 2074 e 27383 são ints armazenados internamente no mesmo número de bytes, mas eles são campos de tamanhos diferentes quando exibidos na tela ou gravados em um arquivo como texto.
- Portanto, o acesso sequencial com fprintf e fscanf normalmente não é usado para atualizar registros no local.
- Em vez disso, o arquivo inteiro é, normalmente, regravado.

# Leitura de dados de um arquivo de acesso sequencial

- **Para fazer a mudança de nome no exemplo anterior, os registros antes de 300 White 0.00 nesse arquivo de acesso sequencial seriam copiados para um novo arquivo, o novo registro seria gravado e os registros após 300 White 0.00 seriam copiados para o novo arquivo.**
- **Atualizar um registro requer o processamento de cada registro do arquivo.**

# Arquivos de acesso aleatório

- Como já dissemos, os registros em um arquivo criado com a função de saída formatada fprintf não têm necessariamente o mesmo tamanho.
- Contudo, registros individuais de um arquivo de acesso aleatório normalmente têm tamanhos fixos e podem ser acessados diretamente (e, portanto, rapidamente), sem pesquisa por outros registros.
- Isso torna os arquivos de acesso aleatório apropriados para sistemas de reservas aéreas, sistemas bancários, sistemas de ponto de venda e outros tipos de sistemas de processamento de transação que exigem acesso rápido a dados específicos.

# Arquivos de acesso aleatório

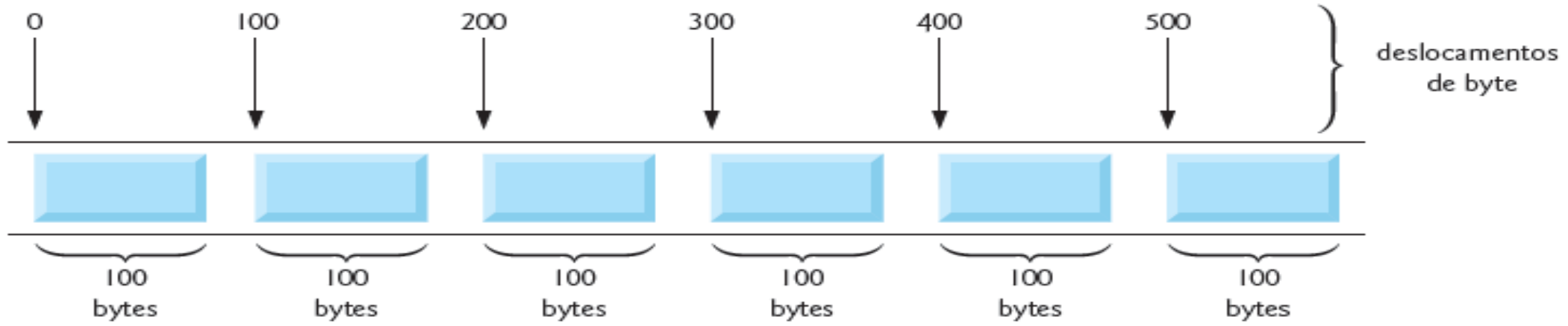


Figura 11.10 ■ A visão da linguagem em C de um arquivo de acesso aleatório.

# Arquivos de acesso aleatório

- Existem outras maneiras de implementar arquivos de acesso aleatório, mas limitaremos nossa discussão a essa técnica direta usando registros de tamanho fixo.
- Como todos os registros em um arquivo de acesso aleatório normalmente têm o mesmo tamanho, o local exato de um registro em relação ao início do arquivo pode ser calculado como uma função da chave de registro.
- Logo veremos como isso facilita o acesso imediato a registros específicos, até mesmo em arquivos grandes.
- A Figura 11.10 ilustra um modo de implementação de um arquivo de acesso aleatório.
- Esse arquivo é como um trem de carga com muitos vagões — alguns vazios e alguns com carga.

# Arquivos de acesso aleatório

- **Registros de tamanho fixo permitem que os dados sejam inseridos em um arquivo de acesso aleatório sem destruir outros dados do arquivo.**
- **Os dados armazenados anteriormente também podem ser atualizados ou excluídos sem que seja necessário regravar o arquivo inteiro.**

# Criação de um arquivo de acesso aleatório

- A função fwrite ransfere para um arquivo um número especificado de bytes a partir de um local especificado na memória.
- Os dados são gravados a partir do local no arquivo indicado pelo ponteiro de posição do arquivo.
- A função fread transfere um número especificado de bytes do local no arquivo especificado pelo ponteiro de posição do arquivo para uma área na memória que começa com o endereço especificado.

# Criação de um arquivo de acesso aleatório

- Agora, ao gravar um inteiro, em vez de usar
  - fprintf( fPtr, "%d", number );

que poderia imprimir um único dígito, ou até 11 dígitos (10 dígitos mais um sinal, cada um exigindo 1 byte de armazenamento), para um inteiro de 4 bytes, podemos usar

  - fwrite( &number, sizeof( int ), 1, fPtr );

que sempre grava 4 bytes (ou 2 bytes em um sistema com inteiros de 2 bytes) a partir de uma variável number para o arquivo representado por fPtr (explicaremos o argumento 1 em breve).

# Criação de um arquivo de acesso aleatório

- **Mais à frente, fread pode ser usado para ler 4 desses bytes em uma variável inteira number.**
- **Embora fread e fwrite leiam e escrevam dados, por exemplo, inteiros, em tamanho fixo, em vez de em formato de tamanho variável, os dados que eles tratam são  rocessados no formato de 'dados brutos' do computador (ou seja, bytes de dados) em vez de em formato de texto compreensível ao humano de printf e scanf.**
- **Como a representação 'bruta' dos dados depende do sistema, os 'dados brutos' podem não ser legíveis em outros sistemas, ou por programas produzidos por outros compiladores ou com outras opções de compilador.**

# Criação de um arquivo de acesso aleatório

- As funções fwrite e fread são capazes de ler e gravar arrays de dados, do disco e para o disco.
- O terceiro argumento de fread e fwrite é o número de elementos no array que devem ser lidos do disco ou gravados no disco.
- A chamada de função fwrite do exemplo anterior grava um único inteiro no disco, de modo que o terceiro argumento é 1 (como se um elemento de um array estivesse sendo gravado).
- Os programas de processamento de arquivo raramente gravam um único campo em um arquivo.
- Normalmente, eles gravam uma struct de cada vez, como mostraremos nos exemplos a seguir.

# Criação de um arquivo de acesso aleatório

- **Considere o seguinte problema:**
  - **Crie um sistema de processamento de crédito capaz de armazenar até 100 registros de tamanho fixo. Cada registro deve consistir em um número de conta que será usado como chave de registro, um sobrenome, um nome e um saldo. O programa resultante deverá ser capaz de atualizar uma conta, inserir um novo registro de conta, excluir uma conta e listar todos os registros de conta em um arquivo de texto formatado para impressão. Use um arquivo de acesso aleatório.**
- **As próximas seções introduzirão as técnicas necessárias para a criação do programa de processamento de crédito.**

# Criação de um arquivo de acesso aleatório

```c
/* Fig. 11.11: fig11_11.c
   Criando um arquivo de acesso aleatório sequencialmente */
#include <stdio.h>

/* definição da estrutura clientData */
struct clientData {
   int acctNum;   /* número da conta */
   char lastName[ 15 ]; /* sobrenome da conta */
   char firstName[ 10 ]; /* nome da conta */
   double balance;   /* saldo da conta */
}; /* fim da estrutura clientData */

int main( void )
{
   int i; /* contador usado para contar de 1-100 */

   /* cria clientData com informações padrão */
   struct clientData blankClient = { 0, "", "", 0.0 };

   FILE *cfPtr; /* ponteiro de arquivo credit.dat */

   /* fopen abre o arquivo; sai se não puder abrir arquivo */
   if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
      printf( "Arquivo não pode ser aberto.\n" );
   } /* fim do if */
   else {
      /* envia 100 registros vazios para arquivo */
      for ( i = 1; i <= 100; i++ ) {
      fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
      } /* fim do for */

      fclose ( cfPtr ); /* fclose fecha o arquivo */
   } /* fim do else */

   return 0; /* indica conclusão bem-sucedida */
} /* fim do main */
```

Figura 11.11 ■ Criação de um arquivo sequencial de acesso aleatório.

# Criação de um arquivo de acesso aleatório

- A Figura 11.11 mostra como abrir um arquivo de acesso aleatório, definir um formato de registro usando uma struct, gravar dados no disco e fechar o arquivo.
- Usando a função fwrite, esse programa inicializará os 100 registros do arquivo “credit.dat” com structs vazias.
- Cada struct vazia contém 0 para o número de conta, "" (a string vazia) para o sobrenome, "" para o nome e 0.0 para o saldo.
- O arquivo será inicializado dessa maneira para criar espaço no disco em que ele será armazenado e para que seja possível determinar se um registro contém dados.

# Criação de um arquivo de acesso aleatório

- A função fwrite grava um bloco (número específico de bytes) de dados em um arquivo.
- Em nosso programa, a linha 29 faz com que a estrutura blankClient de tamanho sizeof( struct clientData ) seja gravada no arquivo apontado por cfPtr.
- O operador sizeof retorna o tamanho de seu operando entre parênteses (nesse caso, struct clientData).
- O operador sizeof retorna um inteiro sem sinal e pode ser usado para determinar o tamanho, em bytes, de qualquer tipo de dado ou expressão.

# Criação de um arquivo de acesso aleatório

- Por exemplo, sizeof(int) pode ser usado para determinar se um inteiro é armazenado em 2 ou 4 bytes em um computador.
- A função fwrite pode realmente ser usada para gravar vários elementos de um array (vetor) de objetos.
- Para gravar vários elementos do array, forneça, na chamada a fwrite, um ponteiro para um array como primeiro argumento, e o número de elementos a serem gravados como terceiro argumento.
- Na instrução anterior, fwrite foi usado para gravar um único objeto que não era um elemento de array.

# Escrita aleatória de dados em um arquivo de acesso aleatório

```
1  /* Fig. 11.12: fig11_12.c
2     Gravando em um arquivo de acesso aleatório */
3  #include <stdio.h>
4
5  /* definição da estrutura clientData */
6  struct clientData {
7     int acctNum;   /* número da conta */
8     char lastName[ 15 ]; /* sobrenome da conta */
9     char firstName[   10 ]; /* nome da conta */
10    double balance;   /* saldo da conta */
11 }; /* fim da estrutura clientData */
12
13 int main( void )
14 {
15    FILE *cfPtr; /* ponteiro do arquivo credit.dat */
16
17    /* cria clientData com informação-padrão */
18    struct clientData client = { 0, "", "", 0.0 };
19
20    /* fopen abre o arquivo; sai se não puder abrir arquivo */
21    if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
22       printf( "Arquivo não pode ser aberto.\n" );
23    } /* fim do if */
24    else {
25       /* exige que usuário especifique número de conta */
26       printf( "Digite número de conta"
27             ( 1 a 100, 0 para encerrar entrada )\n? " );
28       scanf( "%d", &client.acctNum );
29
```

# Escrita aleatória de dados em um arquivo de acesso aleatório

```
30          /* usuário entra informações, copiadas para o arquivo*/
31          while ( client.acctNum != 0 ) {
32             /* usuário digita sobrenome, nome e saldo */
33             printf( "Digite sobrenome, nome e saldo\n? " );
34
35             /* define valor de nome, sobrenome e saldo do registro */
36             fscanf( stdin, "%s%s%lf", client.lastName,
37                client.firstName, &client.balance );
38
39             /* busca posição no arquivo para registro especificado pelo usuário */
40             fseek( cfPtr, ( client.acctNum - 1 ) *
41                sizeof( struct clientData ), SEEK_SET );
42
43             /* grava informação especificada pelo usuário no arquivo */
44             fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
45
46             * permite que usuário informe outro número de conta */
47             printf( "Enter account number\n? " );
48             scanf( "%d", &client.acctNum );
49          } /* fim do while */
50
51          fclose( cfPtr ); /* fclose fecha o arquivo */
52       } /* fim do else */
53
54       return 0; /* indica conclusão bem-sucedida */
55    } /* fim do main */
```

Figura 11.12 ■ Gravação aleatória de dados em um arquivo de acesso aleatório.

# Criação de um arquivo de acesso aleatório

- **A gravação de um único objeto é equivalente a gravar um elemento de um array, por isso o 1 na chamada a fwrite.**
- **[*Nota:* as figuras 11.12, 11.15 e 11.16 usam o arquivo de dados criado na Figura 11.11, por isso você precisa executar o programa da Figura 11.11 antes dos programas nas figuras 11.12, 11.15 e 11.16.]**

# Escrita aleatória de dados em um arquivo de acesso aleatório

- O programa da Figura 11.12 grava dados no arquivo "credit.dat".
- Ele usa a combinação das funções fseek e fwrite para armazenar dados em posições específicas no arquivo.
- A função fseek inicializa o ponteiro de posição do arquivo em uma posição específica no arquivo, e depois fwrite grava os dados.
- Um exemplo de execução é mostrado na Figura 11.13.

# Escrita aleatória de dados em um arquivo de acesso aleatório

- As linhas 40-41 posicionam o ponteiro de posição do arquivo para o arquivo referenciado por cfPtr para o local de byte calculado por ( client.accountNum - 1 ) * sizeof( struct clientData ).
- O valor dessa expressão é chamado de offset ou deslocamento.
- Como o número de conta está entre 1 e 100, mas as posições de byte no arquivo começam com 0, 1 é subtraído do número de conta quando o cálculo do local de byte do registro é feito.

# Escrita aleatória de dados em um arquivo de acesso aleatório

```
Digite número de conta ( 1 a 100, 0 para encerrar entrada )
? 37
Digite sobrenome, nome e saldo
? Barker Doug 0.00
Digite número de conta
? 29
Digite sobrenome, nome e saldo
? Brown Nancy -24.54
Digite número de conta
? 96
Digite sobrenome, nome e saldo
? Stone Sam 34.98
Digite número de conta
? 88
Digite sobrenome, nome e saldo
? Smith Dave 258.34
Digite número de conta
? 33
Digite sobrenome, nome e saldo
? Dunn Stacey 314.33
Digite número de conta
? 0
```

Figura 11.13 ■ Exemplo de execução do programa na Figura 11.12.

# Escrita aleatória de dados em um arquivo de acesso aleatório

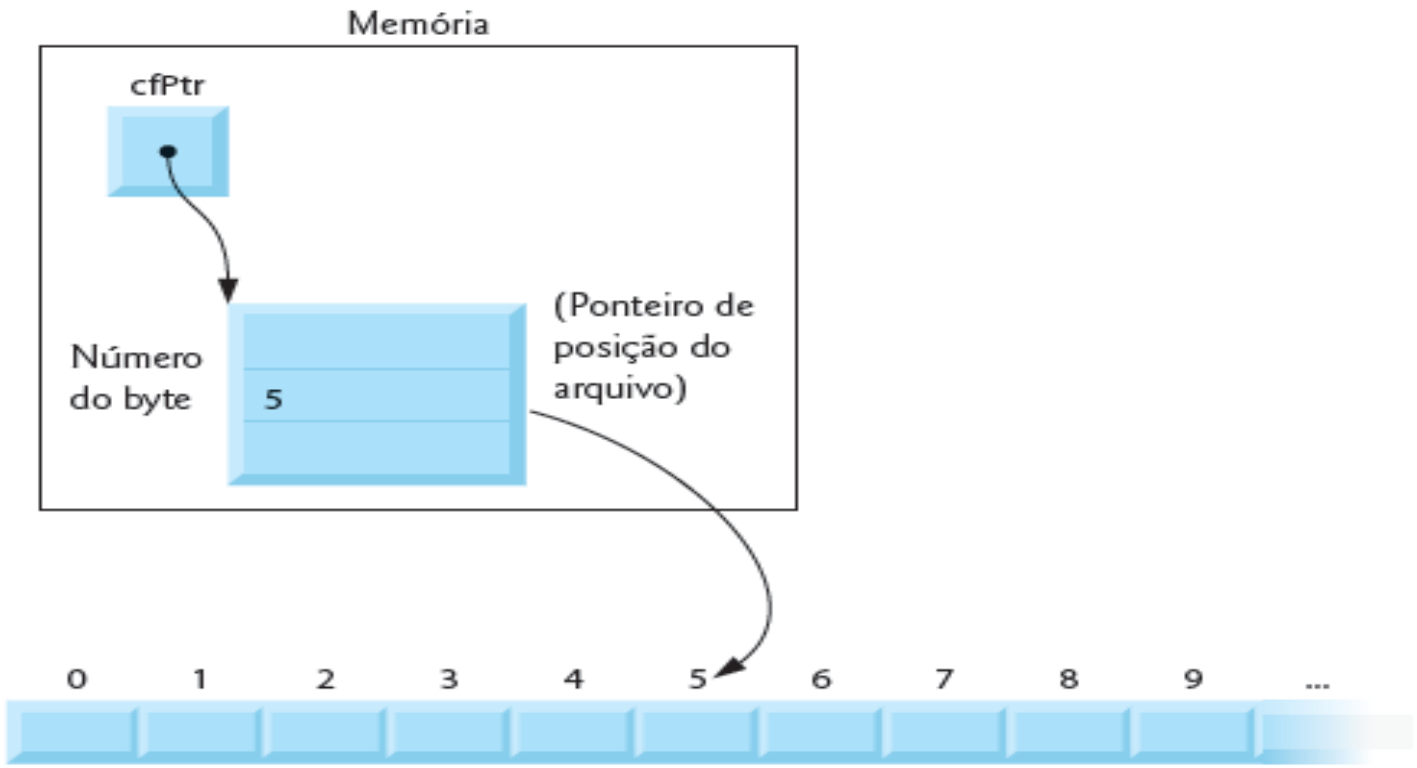


Figura 11.14 ■ Ponteiro de posição do arquivo indicando um deslocamento de 5 bytes a partir do início do arquivo.

# Escrita aleatória de dados em um arquivo de acesso aleatório

- Assim, para o registro 1, o ponteiro de posição do arquivo é definido como o byte 0 do arquivo.
- A constante simbólica SEEK_SET indica que o ponteiro de posição do arquivo está posicionado em relação ao início do arquivo pela quantidade de offset.
- Como as instruções acima indicam, uma busca pelo número 1 de conta nos conjuntos de arquivos no ponteiro de posição de arquivo ao seu início, pois o local de byte calculado é 0.
- A Figura 11.14 ilustra o ponteiro do arquivo referenciando uma estrutura FILE na memória.
- O ponteiro de posição do arquivo nesse diagrama indica que o próximo byte a ser lido ou gravado está a 5 bytes do início do arquivo.

# Escrita aleatória de dados em um arquivo de acesso aleatório

- O protótipo de função para fseek é
  - int fseek( FILE *stream, long int offset,
    int whence );
- onde offset é o número de bytes a ser buscado a partir do local whence no arquivo apontado por stream.
- O argumento whence (de onde) pode ter um dentre três valores — SEEK_SET, SEEK_CUR ou SEEK_END (todos definidos em <stdio.h>) — indicando o local no arquivo onde a busca começa.

# Escrita aleatória de dados em um arquivo de acesso aleatório

- SEEK_SET indica que a busca começa no início do arquivo; SEEK_CUR indica que a busca começa no local atual no arquivo; e SEEK_END indica que a busca começa no final do arquivo.
- Para simplificar, os programas neste capítulo não realizam verificação de erro.
- Se quiser determinar se funções como fscanf (linhas 36–37), fseek (linhas 40–41) e fwrite (linha 44) operam corretamente, você poderá verificar seus valores de retorno.
- A função fscanf retorna o número de itens de dados lidos com sucesso ou o valor EOF, se ocorreu um problema durante a leitura dos dados.

# Escrita aleatória de dados em um arquivo de acesso aleatório

- A função fseek retorna um valor diferente de zero, se a operação de busca não puder ser realizada.
- A função fwrite retorna o número de itens que ela escreveu com sucesso.
- Se esse número for menor que o terceiro argumento na chamada da função, então houve um erro de gravação.

# Leitura de dados de um arquivo de acesso aleatório

- A função fread lê um número especificado de bytes a partir de um arquivo para a memória.
- Por exemplo,
  - fread( &client, sizeof( struct clientData ), 1, cfPtr );

  lê o número de bytes determinado por sizeof( struct clientData ) do arquivo referenciado por cfPtr, e armazena os dados na estrutura client.
- Os bytes são lidos a partir do local no arquivo especificado pelo ponteiro de posição do arquivo.

# Leitura de dados de um arquivo de acesso aleatório

- A função fread pode ser usada para ler vários elementos de array de tamanho fixo, oferecendo um ponteiro ao array em que os elementos serão armazenados e indicando o número de elementos a serem lidos.
- A instrução no exemplo especifica que um elemento deverá ser lido.
- Para ler mais de um elemento, especifique o número de elementos no terceiro argumento da instrução fread.
- A função fread retorna o número de itens que ela leu com sucesso.

# Leitura de dados de um arquivo de acesso aleatório

```
1  /* Fig. 11.15: fig11_15.c
2     Lendo um arquivo de acesso aleatório sequencialmente */
3  #include <stdio.h>
4
5  /* definição de estrutura clientData */
6  struct clientData {
7     int acctNum; /* número de conta */
8     char lastName[ 15 ]; /* sobrenome da conta */
9     char firstName[ 10 ]; /* nome da conta */
10    double balance; /* saldo da conta */
11 }; /* fim da estrutura clientData */
12
13 int main( void )
14 {
15    FILE *cfPtr; /* ponteiro de arquivo credit.dat */
16
17    /* cria clientData com informação default */
18    struct clientData client = { 0, "", "", 0.0 };
19
20    /* fopen abre o arquivo; sai se arquivo não puder ser aberto */
21    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
22       printf( "Arquivo não pode ser aberto.\n" );
23    } /* fim do if */
24    else {
25       printf( "%-6s%-16s%-11s%10s\n", "Conta", "Sobrenome",
26          "Nome", "Saldo" );
```

Figura 11.15 ■ Leitura de um arquivo de acesso aleatório sequencialmente. (Parte 1 de 2.)

# Leitura de dados de um arquivo de acesso aleatório

```
27
28          /* lê todos os registros do arquivo (até eof) */
29          while ( !feof( cfPtr ) ) {
30             fread( &client, sizeof( struct clientData ), 1, cfPtr );
31
32             /* mostra registro */
33             if ( client.acctNum != 0 ) {
34             printf( "%-6d%-16s%-11s%10.2f\n",
35                client.acctNum, client.lastName,
36                client.firstName, client.balance );
37             } /* fim do if */
38          } /* fim do while */
39
40          fclose( cfPtr ); /* fclose fecha o arquivo */
41       } /* fim do else */
42
43       return 0; /* indica conclusão bem-sucedida */
44    } /* fim do main */
```

```
Conta      Sobrenome       Nome         Saldo
29         Brown           Nancy         -24.54
33         Dunn            Stacey        314.33
37         Barker          Doug            0.00
88         Smith           Dave          258.34
96         Stone           Sam            34.98
```

Figura 11.15 ■ Leitura de um arquivo de acesso aleatório sequencialmente. (Parte 2 de 2.)

# Leitura de dados de um arquivo de acesso aleatório

- Se esse número for menor que o terceiro argumento na chamada da função, então houve um erro de leitura.
- A Figura 11.15 lê seqüencialmente cada registro no arquivo "credit.dat", determina se todos os registros contêm dados e exibe os dados formatados para os registros que contêm dados.
- A função feof determina quando o final do arquivo é alcançado, e a função fread transfere dados do disco para a estrutura clientData client.

# Estudo de caso: programa de processamento de transações

- Agora, apresentaremos um programa de porte para processamento de transações usando arquivos de acesso aleatório.
- O programa mantém informações de contas bancárias, atualiza contas existentes, acrescenta novas contas, elimina contas e armazena uma lista formatada de todas as contas-correntes em um arquivo de texto para impressão.
- Supomos que o programa da Figura 11.11 tenha sido executado para podemos criar o arquivo credit.dat.

# Estudo de caso: programa de processamento de transações

- O programa tem cinco opções.
- A opção 1 chama a função textFile (linhas 65-95) para armazenar uma lista formatada de todas as contas em um arquivo de texto chamado accounts.txt, que pode ser impresso mais tarde.
- A função usa fread e técnicas de acesso sequencial ao arquivo usadas no programa da Figura 11.15.

# Estudo de caso: programa de processamento de transações

- A opção 2 chama a função updateRecord (linhas 98-142) para atualizar uma conta.
- A função só atualizará um registro que já existe, de modo que a função verifica primeiro se o registro especificado pelo usuário está vazio.
- O registro é lido para a estrutura client com fread, depois o membro acctNum é comparado com 0.
- Se ele for 0, o registro não conterá informações, e uma mensagem será impressa indicando que o registro está vazio.
- Depois, as escolhas de menu serão exibidas.
- Se o registro contiver informações, a função updateRecord lerá o valor da transação, calculará o novo saldo e regravará o registro no arquivo.

# Estudo de caso: programa de processamento de transações

- A opção 3 chama a função newRecord (linhas 179-218) para acrescentar uma nova conta ao arquivo.
- Se o usuário digita um número de conta para uma conta existente, newRecord exibe uma mensagem de erro informando que o registro já contém informações, e as escolhas do menu são apresentadas novamente.
- Essa função usa o mesmo processo do programa da Figura 11.12 para acrescentar uma nova conta.

# Estudo de caso: programa de processamento de transações

- A opção 4 chama a função deleteRecord (linhas 145-176) para excluir um registro do arquivo.
- A exclusão é realizada com o pedido ao usuário que informe o número da conta e com a reinicialização do registro.
- Se a conta não possui informações, deleteRecord mostra uma mensagem de erro informando que a conta não existe.
- A opção 5 finaliza a execução do programa.
- O programa aparece na Figura 11.16.
- O arquivo"credit.dat" é aberto para atualização (leitura e gravação) com o modo "rb+".

# Estudo de caso: programa de processamento de transações

```c
/* Fig. 11.16: fig11_16.c
   Esse programa lê um arquivo de acesso aleatório sequencialmente,
   atualiza dados já gravados no arquivo, cria novos dados para o
   arquivo e exclui dados que previamente existiam no arquivo. */
#include <stdio.h>

/* definição da estrutura clientData */
struct clientData {
   int acctNum; /* número da conta */
   char lastName[ 15 ]; /* sobrenome da conta */
   char firstName[ 10 ]; /* nome da conta */
   double balance; /* saldo da conta */
}; /* fim da estrutura clientData */

/* protótipos */
int enterChoice( void );
void textFile( FILE *readPtr );
void updateRecord( FILE *fPtr );
void newRecord( FILE *fPtr );
void deleteRecord( FILE *fPtr );

int main( void )
{
   FILE *cfPtr; /* ponteiro de arquivo credit.dat */
   int choice; /* escolha do usuário */

   /* fopen abre o arquivo; sai se arquivo não puder ser aberto */
   if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
      printf( "Arquivo não pode ser aberto.\n" );
   } /* fim do if */
   else {
      /* permite que usuário especifique a ação */
      while ( ( choice = enterChoice() ) != 5 ) {
         switch ( choice ) {
            /* cria arquivo de texto pelo arquivo de registros */
            case 1:
```

# Estudo de caso: programa de processamento de transações

```
37                textFile( cfPtr );
38                break;
39             /* atualiza registro */
40             case 2:
41                updateRecord( cfPtr );
42                break;
43             /* cria registro */
44             case 3:
45                newRecord( cfPtr );
46                break;
47             /* exclui registro existente */
```

Figura 11.16 ■ Programa de contas bancárias. (Parte 1 de 4.)

# Estudo de caso: programa de processamento de transações

```
48              case 4:
49                 deleteRecord( cfPtr );
50                 break;
51              /* mostra mensagem se usuário não selecionou escolha válida */
52              default:
53                 printf( "Escolha incorreta\n" );
54                 break;
55           } /* fim do switch */
56        } /* fim do while */
57
58        fclose( cfPtr ); /* fclose fecha o arquivo */
59     } /* fim do else */
60
61     return 0; /* indica conclusão bem-sucedida */
62  } /* fim do main */
63
64  /* cria arquivo de texto formatado para impressão */
65  void textFile( FILE *readPtr )
66  {
67     FILE *writePtr; /* ponteiro de arquivo accounts.txt */
68
69     /* cria clientData com informação default */
70     struct clientData client = { 0, "", "", 0.0 };
71
72     /* fopen abre o arquivo; sai se arquivo não puder ser aberto */
73     if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
74        printf( "Arquivo não pode ser aberto.\n" );
75     } /* fim do if */
76     else {
77        rewind( readPtr ); /* define ponteiro para início do arquivo */
78        fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
79           "Conta", "Sobrenome", "Nome","Saldo" );
80
81        /* copia todos os registros do arquivo de acesso aleatório para o arquivo de texto */
82        while ( !feof( readPtr ) ) {
```

# Estudo de caso: programa de processamento de transações

```
83            fread( &client, sizeof( struct clientData ), 1, readPtr );
84
85            /* grava único registro no arquivo de texto */
86            if ( client.acctNum != 0 ) {
87               fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
88                  client.acctNum, client.lastName,
89                  client.firstName, client.balance );
90            } /* fim do if */
91         } /* fim do while */
92
93         fclose( writePtr ); /* fclose fecha o arquivo */
94      } /* fim do else */
95   } /* fim da função textFile */
96
97   /* atualiza saldo no registro */
98   void updateRecord( FILE *fPtr )
99   {
100      int account; /* número de conta */
101      double transaction; /* valor da transação */
102
103      /* cria clientData sem informações */
104      struct clientData client = { 0, "", "", 0.0 };
105
106      /* obtém número de conta para atualizar */
107      printf( "Digite conta a atualizar ( 1 - 100 ): " );
108      scanf( "%d", &account );
109
110      /* move ponteiro de arquivo para registro correto no arquivo */
111      fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
```

Figura 11.16 ■ Programa de contas bancárias. (Parte 2 de 4.)

# Estudo de caso: programa de processamento de transações

```
112         SEEK_SET );
113
114      /* lê registro do arquivo */
115      fread( &client, sizeof( struct clientData ), 1, fPtr );
116
117      /* exibe erro se a conta não existir */
118      if ( client.acctNum == 0 ) {
119         printf( "Conta #%d não possui informações.\n", account );
120      } /* fim do if */
121      else { /* atualiza registro */
122         printf( "%-6d%-16s%-11s%10.2f\n\n",
123            client.acctNum, client.lastName,
124            client.firstName, client.balance );
125
126         /* solicita do usuário o valor da transação */
127         printf( "Digite cobrança ( + ) ou pagamento ( - ): " );
128         scanf( "%lf", &transaction );
129         client.balance += transaction; /* atualiza saldo do registro */
130
131         printf( "%-6d%-16s%-11s%10.2f\n",
132            client.acctNum, client.lastName,
133            client.firstName, client.balance );
134
135         /* move ponteiro de arquivo para registro correto */
136         fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
137         SEEK_SET );
138
139         /* grava registro atualizado sobre antigo registro no arquivo */
140         fwrite( &client, sizeof( struct clientData ), 1, fPtr );
141      } /* fim do else */
142   } /* fim da função updateRecord */
143
144   /* exclui um registro existente */
145   void deleteRecord( FILE *fPtr )
146   {
```

# Estudo de caso: programa de processamento de transações

```
147     struct clientData client; /* armazena registro lido do arquivo */
148     struct clientData blankClient = { 0, "", "", 0 }; /* cliente em branco */
149
150     int accountNum; /* número de conta */
151
152     /* obtém número de conta a excluir */
153     printf( "Digite número de conta a excluir ( 1 - 100 ): " );
154     scanf( "%d", &accountNum );
155
156     /* move ponteiro de arquivo para registro correto */
157     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
158        SEEK_SET );
159
160     /* lê registro do arquivo */
161     fread( &client, sizeof( struct clientData ), 1, fPtr );
162
163     /* exibe erro se o registro não existir */
164     if ( client.acctNum == 0 ) {
165        printf( "Conta %d não existe.\n", accountNum );
166     } /* fim do if */
167     else { /* exclui registro */
168        /* move ponteiro de arquivo para registro correto */
169        fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
170           SEEK_SET );
171
172        /* substitui registro existente por registro em branco */
173        fwrite( &blankClient,
174           sizeof( struct clientData ), 1, fPtr );
175     } /* fim do else */
```

Figura 11.16 ■ Programa de contas bancárias. (Parte 3 de 4.)

# Estudo de caso: programa de processamento de transações

```
176  } /* fim da função deleteRecord */
177
178  /* cria e insere registro */
179  void newRecord( FILE *fPtr )
180  {
181     /* cria clientData com informações-padrão */
182     struct clientData client = { 0, "", "", 0.0 };
183
184     int accountNum; /* número de conta */
185
186     /* obtém número de conta a ser criada */
187     printf( "Digite novo número de conta ( 1 - 100 ): " );
188     scanf( "%d", &accountNum );
189
190     /* move ponteiro de arquivo para registro correto no arquivo */
191     fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
192        SEEK_SET );
193
194     /* lê registro do arquivo */
195     fread( &client, sizeof( struct clientData ), 1, fPtr );
196
197     /* mostra erro se a conta já existir */
198     if ( client.acctNum != 0 ) {
199        printf( "Conta #%d já contém informações.\n",
200           client.acctNum );
201     } /* fim do if */
202     else { /* cria registro */
203        /* usuário digita sobrenome, nome e saldo */
204        printf( "Digite sobrenome, nome, saldo\n? " );
205        scanf( "%s%s%lf", &client.lastName, &client.firstName,
206           &client.balance );
207
208        client.acctNum = accountNum;
209
```

# Estudo de caso: programa de processamento de transações

```
210         /* move ponteiro de arquivo para registro correto */
211         fseek( fPtr, ( client.acctNum - 1 ) *
212            sizeof( struct clientData ), SEEK_SET );
213
214         /* insere registro no arquivo */
215         fwrite( &client,
216            sizeof( struct clientData ), 1, fPtr );
217      } /* fim do else */
218   } /* fim da função newRecord */
219
220   /* permite que usuário insira escolha do menu */
221   int enterChoice( void )
222   {
223      int menuChoice; /* variável para armazenar escolha do usuário */
224
225      /* exibe opções disponíveis */
226      printf( "\nDigite sua escolha\n"
227         "1 - armazena um arquivo de texto formatado de contas, chamado\n"
228         " \"accounts.txt\" para impressão\n"
229         "2 - atualiza uma conta\n"
230         "3 - inclui uma nova conta\n"
231         "4 - exclui uma conta\n"
232         "5 - termina o programa\n? " );
233
234      scanf( "%d", &menuChoice ); /* recebe escolha do usuário */
235      return menuChoice;
236   } /* fim da função enterChoice */
```

Figura 11.16 ■ Programa de contas bancárias. (Parte 4 de 4.)

# Lembre-se

**“ O otimismo é um risco ocupacional da programação: o feedback é o tratamento. ”**

**Kent Beck**