

# Arquivo em C


## Parte 2: Arquivos modo binário

# Conteúdo

- Introdução
  - Dificuldade de atualização em arquivos texto
  - Vantagens com arquivo binário
- Manipulando arquivos binários
  - Abrindo
  - Modo de abertura de arquivo binário
  - Escrevendo/Lendo
  - Posicionando em arquivo binário
  - Fechando
- Exemplo Conta Corrente

# Introdução

Fim de  
arquivo EOF



- Analogia de arquivo modo texto: fita continua...

```
1 jose 10.00 2 maria 23.00 3 cristina 23.00 .....
```

- Dificuldade de atualização em arquivos texto
  - O que aconteceria se alterar o saldo de jose de 10.00 para 1,000.23?

```
1 jose 1,000.00 maria 23.00 3 cristina 23.00 .....
```

- O que aconteceria se alterar o nome de maria para marina?

```
1 jose 10.00 2 mariana3.00 3 cristina 23.00 .....
```

- Quando perceber que haverá necessidade de atualização
  - Arquivo Binário

# Introdução

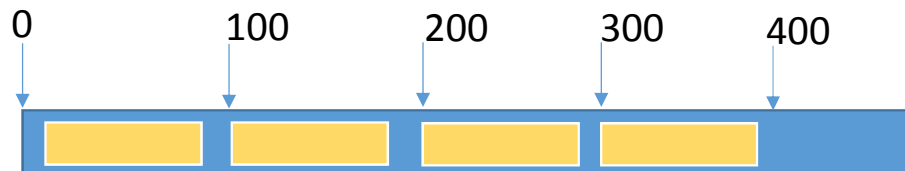
```
unsigned int conta;  
char nome[ 30 ];  
double saldo;
```

1 jose 10.00 2 maria 23.00 3 cristina 23.00 .....

1 joseane 103.00 2 mariana 23.00 3 cristina 230.00 .....

- Quando armazenamos modo texto:
  - cada campo pode variar de tamanho.

- Arquivo modo binário:



 Vagão cada um com 100 bytes

- Vantagens na manipulação com arquivo binário
  - Randômico (rapidez)
  - Facilidade na alteração (inserir dados ou alterar sem destruição).

```
struct Conta {  
    unsigned int conta    // bytes?  
    char nome[ 30 ]      // bytes?  
    double saldo;        // bytes?  
}  
Sempre o mesmo tamanho!!!
```

# Manipulando arquivos binários

- 1ª Abertura de Arquivo:

**fopen**(Arquivo, **MODO**): função que abre o arquivo.

FILE **fopen**( const char \*nome\_arquivo, const char \*modo\_abertura );

**fptr = fopen**("cliente.dat","wb"); // Abertura do arquivo binário para escrita

**fptr = fopen**(" cliente.dat","rb"); // Abertura do arquivo binário para leitura

- O valor de retorno da função **fopen()** é muito importante! Ele é o identificador do fluxo que você abriu e **é só com ele que você conseguirá ler e escrever** no arquivo aberto.
- Se houver um erro na abertura/criação do arquivo, a função retornará o valor **NULL**. O erro geralmente acontece por duas razões:
  - O arquivo não existe, caso tenha sido requisitado para leitura.
  - O usuário atual não tem permissão para abrir o arquivo com o modo de acesso pedido. Por exemplo, o arquivo é somente-leitura, ou está bloqueado para gravação por outro programa, ou pertence a outro usuário e não tem permissão para ser lido por outros.

# Manipulando arquivos binários

Modo	Significado
<b>r</b>	Abre o arquivo somente para leitura. O arquivo deve existir. (O <i>r</i> vem do inglês <i>read</i> , ler)
<b>r+</b>	Abre o arquivo para leitura e escrita. O arquivo deve existir.
<b>w</b>	Abre o arquivo somente para escrita no início do arquivo. Apagará o conteúdo do arquivo se ele já existir, criará um arquivo novo se não existir. (O <i>w</i> vem do inglês <i>write</i> , escrever)
<b>w+</b>	Abre o arquivo para escrita e leitura, apagando o conteúdo pré-existente.
<b>a</b>	Abre o arquivo para escrita no final do arquivo. Não apaga o conteúdo pré-existente. (O <i>a</i> vem do inglês <i>append</i> , adicionar, apender)
<b>a+</b>	Abre o arquivo para escrita no final do arquivo e leitura.

- Em ambientes DOS/Windows, ao ler arquivos binários (por exemplo, programas executáveis ou certos tipos de arquivos de dados), deve-se adicionar o caractere "b" ao final da string de modo (por exemplo, "wb" ou "r+b") para que o arquivo seja lido/gravado corretamente.
- Isso é necessário porque no modo texto (o padrão quando não é adicionado o *b*) ocorrem algumas traduções de caracteres (por exemplo, a terminação de linha "\r\n" é substituída apenas por "\n" na leitura) que poderiam afetar a leitura/gravação dos arquivos binários (indevidamente inserindo ou suprimindo caracteres).

unsigned fwrite(void \*buffer,int numero\_de\_bytes,int count,FILE \*fp);

# ETAPAS: Abertura, **Manipulação** e Fechamento

- 2ª Manipulação de Arquivo (leitura/**escrita**):

- Escrita:

**fwrite** (&VARIÁVEL,TAMANHO,QUANTIDADE,ARQUIVO);

- Escreve no arquivo a variável.
- Recebe quatro argumentos: o endereço da variável, o tamanho em byte da variável, a quantidade de registros e o ponteiro para a estrutura FILE do arquivo.
- Retorna o número de itens escritos.

int **fwrite** (void \*buffer,int numero\_de\_bytes,int count,FILE \*fp);

fwrite( &código, sizeof(int) , 1, fptr);

fwrite( &vazioCliente, sizeof( struct clienteConta ), 1, cfPtr );

```
struct clienteConta {  
    unsigned int contaNum;  
    char nome[ 25 ];  
    double saldo;  
};  
  
struct clienteConta vazioCliente;
```

# ETAPAS: Abertura, **Manipulação** e Fechamento

- 2ª Manipulação de Arquivo (**leitura**/escrita):
- Leitura:

**fread** (VARIÁVEL,TAMANHO,QUANTIDADE,ARQUIVO); ):

- Lê os dados do arquivo para a variável.
- Recebe quatro argumentos: o endereço da variável, o tamanho em byte a ser lido, a quantidade de registros, e o ponteiro para a estrutura FILE do arquivo.
- Retorna o número de itens lidos. Esse valor poderá ser menor que QUANTIDADE se o final do arquivo for atingido ou ocorrer um erro.

```
int fread (void *buffer, int numero_de_bytes, int num_elemt, FILE *fp);
```

Ex:

```
fread(&codigo,sizeof(codigo),1, fptr);
```

```
fread( &cliente, sizeof( struct clienteConta ), 1, cfPtr );
```



# ETAPAS: Abertura, Manipulação e Fechamento

- 3ª Fechamento:

**fclose** (ARQUIVO):

- Fecha o arquivo e esvazia o conteúdo do buffer, garantindo que nenhuma informação seja deixado no buffer, também chamado de descarga ou flushing
- Libera as áreas de comunicação entre o programa e sistema operacional.

```
int fclose( FILE *ponteiro_arquivo);
```

Ex: **fclose**(fptr)

# Escrevendo/Lendo Aleatoriamente (posição) movendo pelo arquivo

Para apontar para um byte específico dentro do arquivo movimenta o indicador de posição. Isto pode ser feito com o uso da função **fseek( )**, cuja sintaxe é:

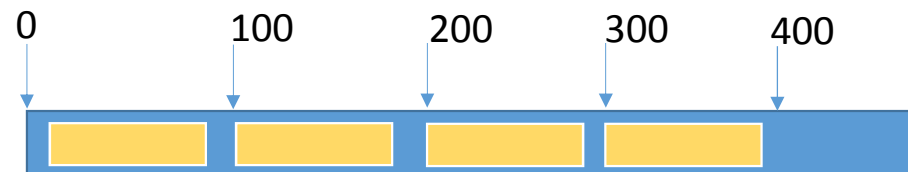
```
fseek(ARQUIVO, NÚMERO_DE_BYTES, ORIGEM);
```

Onde:

ARQUIVO é um ponteiro de arquivo aberto anteriormente,  
NÚMERO\_DE\_BYTES é a quantidade de bytes que o indicador de posição será movimentado e  
ORIGEM é a partir de onde o movimento do indicador de posição iniciará.

ORIGEM deve ser uma das seguintes macros:

- **SEEK\_SET** 0 Início do arquivo
- **SEEK\_CUR** 1 Ponto corrente no arquivo
- **SEEK\_END** 2 Fim do arquivo



```
fseek( fPtr, ( conta - 1 ) * sizeof( struct clienteConta ), SEEK_SET );
```

# Apontando para o início do arquivo (voltar para o início)

- Para apontar para o início do arquivo use a função `rewind( )`

`rewind(ARQUIVO);`

- sendo ARQUIVO um ponteiro de arquivo.
- Outras funções:
- `remove(ARQUIVO);` //remove arquivo
- `rename(ARQUIVO_ANTIGO, ARQUIVO_NOVO);`
- `flush(ARQUIVO);` //esvaziar o conteúdo de um stream

## Criando e ZERANDO Arquivo:

```
1. struct ClienteConta {
2.     int numConta;    // numero da conta
3.     char nome[ 45 ]; // nome
4.     double saldo;    // saldo
5. };

6. int main( void ){
7.
8.     int i; // contador usado para conta de 1-100

9.     struct ClienteConta clienteVazio = { 0, "", 0.0 }; // cria ClienteConta com informação default
10.
11.     FILE *cArquivoPtr; // ponteiro para arquivo contas.dat

12.     if ( (cArquivoPtr = fopen( "contas.dat", "wb" ) ) == NULL ) //abre arquivo para escrita se existe descarta conteúdo
13.         puts( "Arquvio nao pode ser aberto." );
14.     else {
15.
16.         for ( i = 1; i <= 100; ++i ) // gera 100 registros em branco para o arquivo
17.             fwrite( &clienteVazio, sizeof( struct ClienteConta ), 1, cArquivoPtr );
18.
19.         fclose (cArquivoPtr; ); // fclose fecha arquivo
20.
21.     } // fim else

22. } // fim main
```

## Escrevendo apenas 1 estrutura no Arquivo

```
1. struct ClienteConta {
2.     int numConta;           // numero da conta
3.     char nome[ 45 ];        // nome
4.     double saldo;           // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr;
9.     struct ClienteConta cliente;
10.    int numeroConta;

11.
12.    if ( (cArquivoPtr = fopen( "contas.dat", "r+b" )) == NULL ) // abre arquivo para escrita se existe descarta conteúdo
13.        puts( "Arquvio nao pode ser aberto." );
14.    else {
15.        printf( "%s", "Entre numero da conta ( 1 a 100):\n? " );
16.        scanf( "%d", &numeroConta );
17.        fflush(stdin);           //limpa o buffer de entrada do carater enter
18.
19.        fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );

20.        fread( &cliente, sizeof( struct ClienteConta ), 1, cArquivoPtr );
21.
22.        if ( cliente.numConta != 0 )
23.            printf("A conta de numero: %d ja existe", numeroConta );
24.        else{
25.            printf( "%s", "Entre nome:\n? " );
26.            gets(cliente.nome);
27.            printf( "%s", "\nEntre com o saldo:\n? " );
28.            scanf("%lf",&cliente.saldo);
29.
30.            cliente.numConta = numeroConta;
31.
32.            fseek( cArquivoPtr, ( cliente.numConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET ); // seta posicao para registro especifico

33.            fwrite( &cliente, sizeof( struct ClienteConta ), 1, cArquivoPtr ); //escreve a estrutura com o dados para a posição no arquivo
34.        }
35.        fclose( cArquivoPtr );
36.    } // fim else
37. } // fim main
```

## Lendo estrutura do Arquivo randomicamente p/ Tela

```
1. struct clienteConta {
2.     int contaNum;           // numero da conta
3.     char nome[ 45 ];        // nome
4.     double saldo;          // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr;
9.     struct ClienteConta cliente;

10.    if ( (cArquivoPtr = fopen( "contas.dat", "rb" ) ) == NULL ) {
11.        puts( "Arquivo nao pode ser aberto." );
12.    }
13.    else {
14.        printf( "%-6s%-27s%10s\n", "Conta", "Nome", "Saldo" ); //cabecalho

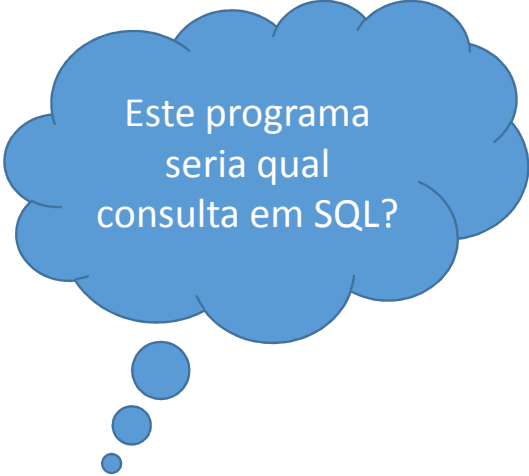
15.        while ( !feof ( cArquivoPtr ) ) { // le todos registros do arquivo ate eof

16.            fread( &cliente, sizeof ( struct ClienteConta ), 1, cArquivoPtr );

17.            if ( cliente.numConta != 0 )
18.                printf( "%-6d%-27s%10.2f\n", cliente.numConta, cliente.nome, cliente.saldo );
19.
20.        }

21.        fclose( cArquivoPtr ); // fclose fecha arquivo
22.
23.    } // fim else

    } // fim main
```



Este programa  
seria qual  
consulta em SQL?

## Lendo estrutura do Arquivo randomicamente e escrevendo modo texto.

```
1. struct clienteConta {
2.     int contaNum;           // numero da conta
3.     char nome[ 45 ];       // nome
4.     double saldo;         // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr; // leitura
9.     FILE *cEscritaPtr; // escrita modo texto
10.    struct ClienteConta cliente;

11.    if ( (cArquivoPtr = fopen( "contas.dat", "rb" ) ) == NULL ) {
12.        puts( "Arquivo nao pode ser aberto." );
13.        return;
14.    }

15.
16.    if ( (cArquivoPtr = fopen( "contas.txt", "w" ) ) == NULL )
17.        puts( "Arquivo nao pode ser aberto." );
18.    else {
19.        fprintf(*cEscritaPtr, "%-6s%-27s%10s\n", "Conta", "Nome", "Saldo" ); //cabecalho

20.        while ( !feof ( cArquivoPtr ) ) { // le todos registros do arquivo ate eof

21.            fread( &cliente, sizeof ( struct ClienteConta ), 1, cArquivoPtr );

22.            if ( cliente.numConta != 0 )
23.                fprintf(cEscritaPtr, "%-6d%-27s%10.2f\n", cliente.numConta, cliente.nome, cliente.saldo );

24.        }

25.
26.        fclose( cArquivoPtr ); // fclose fecha arquivo
27.    } // fim else

28.    return 0;
    } // fim main
```

## Removendo estrutura do Arquivo randomicamente

```
1. struct clienteConta {
2.     int numConta;           // numero da conta
3.     char nome[ 45 ];        // nome
4.     double saldo;           // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr; // leitura
9.     struct ClienteConta cliente;
10.    struct ClienteConta clienteVazio = { 0, "", 0.00 }; //útil para escrever no lugar do removido

11.    if ( (cArquivoPtr = fopen( "contas.dat", "r+b" ) ) == NULL ) {
12.        puts( "Arquivo nao pode ser aberto." );
13.        return;
14.    }
15.
16.    printf( "%s", "Entre numero da conta para ser removida ( 1 a 100):\n? " );
17.    scanf( "%d", &numeroConta );
18.
19.    fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );
20.    fread( &cliente, sizeof( struct ClienteConta ), 1, cArquivoPtr );

21.    if ( cliente.numConta == 0 )
22.        printf("A conta de numero: %d esta VAZIA", numeroConta );
23.    else{
24.        printf("Dados da conta a ser removida \n");
25.        printf( "%-6d%-27s%10.2f\n", cliente.numConta, cliente.nome, cliente.saldo );

26.        fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );
27.        fwrite( &clienteVazio, sizeof( struct ClienteConta ), 1, cArquivoPtr );
28.
29.        printf("remove conta com sucesso!!!" );
30.    }
31.    fclose( cArquivoPtr ); // fclose fecha arquivo
32.    return 0;
} // fim main
```



## Atualizando saldo: deposito e saque em arquivo

```
1. struct clienteConta {
2.     int numConta;           // numero da conta
3.     char nome[ 45 ];       // nome
4.     double saldo;          // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr;      // leitura
9.     struct ClienteConta cliente;
10.    int numConta;
11.    double transacao;        //valor do saque ou deposito

12.    if ( (cArquivoPtr = fopen( "contas.dat", "r+b" ) ) == NULL ) {
13.        puts( "Arquivo nao pode ser aberto." );
14.        return 0;
15.    }
16.
17.    printf( "%s", "Entre numero da conta para ser removida ( 1 a 100):\n? " );
18.    scanf( "%d", &numeroConta );
19.
20.    fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );
21.    fread( &cliente, sizeof( struct ClienteConta ), 1, cArquivoPtr );

22.    if ( cliente.numConta == 0 )
23.        printf("A conta de numero: %d esta VAZIA", numConta );
24.    else{
25.        printf( "%-6d%-27s%10.2f\n", cliente.numConta, cliente.nome, cliente.saldo );
26.        printf( "%s", "Entre deposito ( + ) or saque ( - ): " );
27.        scanf( "%lf", &transacao );
28.        cliente.saldo += transacao; // atualiza saldo do registro

29.        fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );
30.        fwrite( &clienteVazio, sizeof( struct ClienteConta ), 1, cArquivoPtr );

31.    }
32.    fclose( cArquivoPtr ); // fclose fecha arquivo
33.    return 0;
} // fim main
```

```

1. struct clienteConta {
2.     int numConta;           // numero da conta
3.     char nome[ 45 ];        // nome
4.     double saldo;          // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr;      // leitura
9.     struct ClienteConta cliente;
10.    int numConta;
11.    char nome[45];           //valor do saque ou deposito

12.    if ( (cArquivoPtr = fopen( "contas.dat", "r+b" ) ) == NULL ) {
13.        puts( "Arquivo nao pode ser aberto." );
14.        return 0;
15.    }

16.
17.    printf( "%s", "Entre numero da conta para ser removida ( 1 a 100):\n? " );
18.    scanf( "%d", &numeroConta );
19.    fflush(stdin);

20.
21.    fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );
22.    fread( &cliente, sizeof( struct ClienteConta ), 1, cArquivoPtr );

23.    if ( cliente.numConta == 0 )
24.        printf("A conta de numero: %d esta VAZIA", numConta );
25.    else{
26.        printf( "%-6d%-27s%10.2f\n", cliente.numConta, cliente.nome, cliente.saldo );
27.        printf( "%s", "Entre nome: " );
28.        gets( nome );
29.        strcpy(cliente.nome, nome);    // atualiza nome do registro
30.
31.        printf( "%-6d%-27s%10.2f\n", cliente.numConta, cliente.nome, cliente.saldo );
32.
33.        fseek( cArquivoPtr, ( numeroConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET );
34.        fwrite( &clienteVazio, sizeof( struct ClienteConta ), 1, cArquivoPtr );

35.    }
36.    fclose( cArquivoPtr ); // fclose fecha arquivo
37.    return 0;
} // fim main

```

## Atualizando nome da conta em arquivo

# Referências

- Video Aulas (YouTube)
- Programar em C - Manipulação de Arquivos txt em C / Ler Dados -  
[https://www.youtube.com/watch?v=y\\_euDUgoND8](https://www.youtube.com/watch?v=y_euDUgoND8)
- **Programar em C - Manipulação de Arquivos txt em C / Incluir Dados - Aula 84**  
[https://www.youtube.com/watch?annotation\\_id=annotation\\_238568&feature=iv&src\\_vid=y\\_euDUgoND8&v=USsUSMpNGsM](https://www.youtube.com/watch?annotation_id=annotation_238568&feature=iv&src_vid=y_euDUgoND8&v=USsUSMpNGsM)

Sites sobre manipulação de Arquivos em C

[http://homepages.dcc.ufmg.br/~joaoreis/Site%20de%20tutoriais/c\\_int/arquivos.htm](http://homepages.dcc.ufmg.br/~joaoreis/Site%20de%20tutoriais/c_int/arquivos.htm)

[http://pt.wikibooks.org/wiki/Programar\\_em\\_C/Entrada\\_e\\_saida\\_em\\_arquivos](http://pt.wikibooks.org/wiki/Programar_em_C/Entrada_e_saida_em_arquivos)

<http://www.vivaolinux.com.br/artigo/Manipulando-arquivos-em-C-%28parte-1%29/?pagina=4>

[http://homepages.dcc.ufmg.br/~joaoreis/Site%20de%20tutoriais/c\\_int/arquivos.htm](http://homepages.dcc.ufmg.br/~joaoreis/Site%20de%20tutoriais/c_int/arquivos.htm)

[http://pt.wikibooks.org/wiki/Programar\\_em\\_C/Entrada\\_e\\_saida\\_em\\_arquivos](http://pt.wikibooks.org/wiki/Programar_em_C/Entrada_e_saida_em_arquivos)

<http://www.ime.usp.br/~elo/IntroducaoComputacao/Manipulacao%20de%20arquivo.htm>

# Outros.. curiosidades

## Streams padrão

- Quando um programa em linguagem C é iniciado são abertas três streams: **stdin**, **stdout** e **stderr**.
  - **stdin** define a entrada padrão do sistema, normalmente o teclado.
  - **stdout** define a saída padrão do sistema, normalmente o monitor.
  - **stderr** define a saída padrão dos erros, normalmente também é o monitor.

Estas streams são ponteiros de arquivos e podem ser redirecionadas. Assim, nas funções que você utiliza ponteiros de arquivos para entrada e saída de dados você pode muito bem usar estas streams de modo ao seu programa receber dados do teclado e escrever no monitor. Isto foi mostrado no exemplo da seção anterior na linha;

```
fscanf(stdin,"%s",string); /* lê string do teclado */
```

onde o programa leu a variável string do teclado através da streams padrão stdin.

- Porém esteja consciente que **estas streams** não são variáveis e não podem receber um valor. Ou seja, você não pode abrí-las com fopen.
- Quando o programa é encerrado estas streams são fechadas automaticamente, do mesmo jeito que foram criadas, **você não deve nunca tentar abrí-las ou fechá-las.**

# Variações dos exemplos

- Escreve várias contas até que usuário digite 0
  - escrevecontaMult.c
- RemoveConta confirmando remoção com usuário.
  - removeConta.c

## Escrevendo estrutura no Arquivo randomicamente

```
1. struct ClienteConta {
2.     int numConta;           // numero da conta
3.     char nome[ 45 ];       // nome
4.     double saldo;          // saldo
5. };

6. int main( void ){
7.
8.     FILE *cArquivoPtr;
9.     struct ClienteConta cliente;
10.
11.     if ( (cArquivoPtr = fopen( "contas.dat", "r+b" )) == NULL ) // abre arquivo para escrita se existe descarta conteúdo
12.         puts( "Arquivo nao pode ser aberto." );
13.     else {
14.         printf( "%s", "Entre numero da conta ( 1 a 100, 0 para finalizar )\n? " );
15.         scanf( "%d", &cliente.numConta );
16.         fflush(stdin);           //limpa o buffer de entrada do carater enter
17.
18.         while ( cliente.contaNum != 0 ) {
19.             printf( "%s", "Entre nome:\n? " );
20.             gets(cliente.nome);
21.             printf( "%s", "\nEntre com o saldo:\n? " );
22.             scanf("%lf",&cliente.saldo);
23.
24.             fseek( cArquivoPtr, ( cliente.numConta - 1 ) * sizeof( struct ClienteConta ), SEEK_SET ); // seta posicao para registro especifico
25.
26.             fwrite( &cliente, sizeof( struct ClienteConta ), 1, cArquivoPtr ); //escreve a estrutura com o dados para a posição no arquivo
27.
28.             printf( "%s", "Entre com numero da conta:\n? " );
29.             scanf( "%d", &cliente.numConta );
30.             fflush(stdin);           //limpa o buffer de entrada do carater enter
31.
32.         }
33.
34.         fclose( cArquivoPtr );
35.     } // fim else
36. } // fim main
```