1001350: Estudo de caso: registro de vendas

Jander Moreira*

Sumário

1 Apresentação

Este estudo detalha a função de leitura (`fread`) e apresenta o conceito de posição corrente no arquivo. Introduz as funções `ftell` para guardar a posição atual e `fseek` para alterar a posição corrente.

Códigos fonte

Todos os programas citados neste texto têm seu código fonte disponibilizado.

Os principais programas de interesse estão também com seu código na seção 5.

2 Criação de um arquivo de registros

O primeiro passo, antes de conseguir ler de um arquivo, é criá-lo.

Assim, o programa `cria_arquivo.c` contém o fonte para criar um arquivo de registros.

O contexto desse arquivo é que ele registra, para uma loja, cada venda efetuada. Uma venda específica é caracterizada por três campos: o código do produto, a quantidade de itens vendidos e o valor individual do item.

Para isso foi definido o registro apresentado no Código 1.

Código 1

```
// definicao do registro
```

*Jander Moreira – Universidade Federal de São Carlos – Departamento de Computação – *Rodovia Washington Luis, km 235 – 13565-905 - São Carlos/SP – Brasil* – jander@dc.ufscar.br

```
struct venda {
    int codigo;
    int quantidade;
    double valor;
};
```

O programa permite a criação do arquivo tanto digitando os valores (controle humano) quanto gerando os dados aleatoriamente (descontrole total). Isso é solicitado no início da execução. Na primeira opção, digita-se até cansar[1]; na segunda é solicitada a quantidade de registros que serão gravados no arquivo.

Os Códigos 2 e 3 mostram a criação do arquivo nas duas versões. As funções `leia_item` e `item_aleatorio` apenas retornam um `struct venda` lido ou com dados aleatórios, respectivamente. A função `confirma_continuacao` pergunta ao usuário se ele deseja continuar ou parar.

Código 2

```
// abertura do arquivo e verificacao
FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "wb");
if(arquivo_vendas == NULL){
    perror(NOME_ARQUIVO);
    exit(1);
}
while(confirma_continuacao()){
    struct venda item = leia_item();
    fwrite(&item, sizeof item, 1, arquivo_vendas);
}

// encerramento
fclose(arquivo_vendas);
```

Código 3

```
// abertura do arquivo e verificacao
FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "wb");
if(arquivo_vendas == NULL){
    perror(NOME_ARQUIVO);
    exit(1);
}

printf("Quantidade de registros (0 ou mais): ");
int quantidade;
scanf("%d", &quantidade);
for(int i = 0; i < quantidade; i++){
    struct venda item = item_aleatorio();
    fwrite(&item, sizeof item, 1, arquivo_vendas);
}

// encerramento
fclose(arquivo_vendas);
```

A criação de um arquivo vazio pode ser feita, na opção de digitacao dos dados, encerrando a entrada antes de digitar o primeiro registro. Na opção de geração com dados aleatórios, a quantidade de zero registro cria o arquivo vazio.

[1] O encerramento é feito digitando-se `Ctrl-D` (Linux e MacOS) ou `Ctrl-Z+ENTER` (Windows).

Gravação de registros

Neste código merece destaque a gravação de registros inteiros. Essa é a forma "correta" de fazer a gravação[2].

A escrita é feita pelo comando `fwrite` (destacado no Código 4).

Código 4

```
fwrite(&item, sizeof item, 1, arquivo_vendas);
```

Os parâmetror para a escrita são o endereço de um `struct venda` (a variável `item`), o tamanho dela (`sizeof item`), a quantidade de um único registro e o arquivo para o qual copiar os bytes.

O efeito do comando é a cópia de todos os bytes da variável `item` para o arquivo. O uso do `sizeof` é importante para se ter o tamanho correto do registro sem esforço do programador.

3 Leitura sequencial

Esta seção aborda como, em C, é possível fazer a leitura dos dados do arquivo sem saber, *a priori*, a quantidade de elementos armazenados nele.

O código fonte disponível para esta tarefa está no arquivo `lista_registros.c`

O arquivo é aberto com o modo `"rb"`, ou seja, somente para leitura e usando arquivos binários. Veja o Código 5.

Código 5

```
FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "rb");
if(arquivo_vendas == NULL)
    perror(NOME_ARQUIVO);
else ...
```

As leituras são feitas com o comando `fread`, operando sobre uma variável `item`, declarada como um registro `struct venda` (Código 6).

Código 6

```
fread(&item, sizeof item, 1, arquivo_vendas)
```

Detalhes do `fread`

A definição formal, segundo o manual, da função `fread` está no Código 7.

Código 7

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

São pontos de interesse:

- `ptr` é o endereço destino dos dados;
- `size` é o tamanho de cada item que será copiado;
- `nmemb` é o "numero de membros", ou seja quantos elementos do tamanho especificado serão considerados na cópia;
- `stream` é o descritor do arquivo do qual os bytes serão copiados;
- `size_t` é o tipo de retorno da função.

[2] A gravação campo a campo é possível, mas pode não manter a compatibilidade com a leitura do registro inteiro.

A função `fread` retorna a quantidade de itens que efetivamente foram lidos. Por exemplo, se `nmemb` for 1, como é usado no `lista_registros.c`, a função pode retornar 1, se a leitura foi bem sucedida, ou zero, no caso de não conseguir ler.

Essa característica é usada para verificar se chegou ao fim do arquivo. Uma tentativa de ler depois do último falha e, assim, `fread` retorna zero.

Desta forma, a leitura pode ser verificada por um código como o do Código 8. Nele, a função é chamada dentro do `if` e o valor de retorno é verificado para ver se houve sucesso.

Código 8

```
if(fread(&valor, sizeof valor, 1, arquivo) == 0){
    // falha na leitura
}
else{
    // leitura bem sucedida
}
```

Uso do `fread` no programa

No programa para mostrar os registros, essa verificação é feita em um `while` (Código 9). A variável `item` é um `struct venda` e `numero_registro` é um contador simples.

Código 9

```
int numero_registro = 0;
while(fread(&item, sizeof item, 1, arquivo_vendas) > 0){
    printf("Registro %4d: (%d, %d, %.2lf)\n",
        numero_registro++,
        item.codigo, item.quantidade, item.valor);
}
```

Na condição do `while` o valor de retorno de `fread` é confrontado com zero. Se o valor for maior que zero, então os dados foram lidos do arquivo e podem ser usados. A outra possibilidade é o arquivo ter terminado (i.e., o último registro já ter sido lido) e o retorno do valor zero encerrar a repetição.

`ftell`???

O programa inclui, ainda, uma outra informação importante. Em alguns momentos é usada a função `ftell`, que está "escondida" na chamada a `info_posicao` (Código 10).

Código 10

```
// info_posicao: apresenta mensagem contextual sobre a posicao atual
//  do arquivo
// Input: texto com o contexto e o arquivo (FILE*) como parametros
// Output: a mensagem e a posicao corrente do arquivo
void info_posicao(char *mensagem, FILE *stream){
    printf(">>> %s: byte atual = %lu\n", mensagem, ftell(stream));
}
```

Quando se abre um arquivo no modo `"r"`, o sistema operacional se prepara para que a leitura inicie no primeiro byte do arquivo. Esse byte é o byte zero[3] do fluxo de dados.

Essa "posição onde vai ocorrer a leitura" é comumente chamada *posição corrente* do arquivo. É exatamente esse valor que a função `ftell` retorna.

[3] **Surpresa! A contagem dos bytes comeÃğa em zero!**.

Na primeira execução, logo após o `fopen`, é escrito que `byte atual = 0`, que significa que a posição corrente é igual a zero.

A cada leitura pelo `fread`, são consumidos `sizeof item` do fluxo de dados do arquivo e a posição corrente é atualizada para que coincida com o próximo byte a ser lido. Por exemplo, na compilação feita no Linux 64bits, cada registro é formado por 16 bytes e, desse modo, a cada leitura a posição corrente é incrementada de 16.

Ao ser lido o último registro, a posição corrente é o byte logo após o último do arquivo todo. É nessa situação em que uma nova chamada a `fread` falha e a repetição do `while` termina. Observe o Código 9 desta nova perspectiva.

4 Exemplo: maior venda

O programa apresentado em Código 15 (seção 5) varre o arquivo e apresenta o registro individual com o maior valor no campo quantidade.

O primeiro registro é, inicialmente, selecionado (Código 11). Depois todos os outros são varridos e, caso possuam quantidade maior, é feita a atualização do item selecionado (Código 12).

Código 11

```
struct venda item;
if(fread(&item, sizeof item, 1, arquivo_vendas) == 0)
    printf("Arquivo vazio\n");
else{
    // pega o primeiro registro
    struct venda item_selecao = item;
    ...
```

Código 12

```
// varredura do restante do arquivo
while(fread(&item, sizeof item, 1, arquivo_vendas) > 0)
    if(item.quantidade > item_selecao.quantidade)
        item_selecao = item;
```

Depois da varredura o conteúdo de `item_selecao` é mostrado na tela e arquivo tem que ser fechado com `fclose`.

5 Códigos fonte

No Código 13 está o programa da seção 2, que gera o arquivo de dados.

Código 13

```
/*
  Este programa cria um arquivo de registros de venda de uma dada loja.
  Input: a opcao do modo de operacao do programa (teclado, sendo
    1: digitacao ou 2: geracao aleatoria); para opcao 1 seguem os
    dados dos registros (teclado); para a opcao 2 segue a quantidade
    de registros que sera gerada com valores aleatorios
  Output: cada registro digitado ou gerado aleatoriamente gravado no
    arquivo vendas.dat.

  Jander, 2020

  Cada registro guarda:
    - o codigo do produto (inteiro)
                                                                    ▷
```

```
    - a quantidade vendida (inteiro)
    - o valor de venda (double)
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <errno.h>
#include <assert.h>

// definicao do registro
struct venda {
    int codigo;
    int quantidade;
    double valor;
};

#define NOME_ARQUIVO "vendas.dat"

// prototipos
struct venda leia_item(void);
struct venda item_aleatorio(void);
void leitura(void);
void aleatorio(void);

//
// programa principal
//
int main(void){
    srand((unsigned int)time(NULL)); // aleatorizacao

    printf("Escolha:\n"
        "1) para digitar os valores\n"
        "2) para gerar dados aleatoriamente\n"
        "\topcao: ");
    int opcao;
    scanf("%d", &opcao);
    switch(opcao){
        case 1:
          leitura();
          break;
        case 2:
          aleatorio();
          break;
        default:
          printf("Opcao ignorada\n");
    }

    return 0;
}

// leia_item: leitura dos dados de uma venda do teclado
// Input: os valores para codigo, quantidade e valor (via
//  teclado)
// Output: um registro (struct venda) com os valores digitados
//  no retorno da função
struct venda leia_item(void){
    struct venda item;
```

▷

```
    printf("\nCodigo: ");
    scanf("%d", &item.codigo);
    printf("Quantidade: ");
    scanf("%d", &item.quantidade);
    printf("valor (R$): ");
    scanf("%lf", &item.valor);

    return item;
}

// item_aleatorio: gera um item com dados aleatorios, sem
//  qualquer compromisso com a realidade
// Input: nao existe
// Output: um registro (struct venda) com valores dos
//  campos gerados aleatoriamente
struct venda item_aleatorio(void){
    struct venda item;

    item = (struct venda){
        .codigo = rand() % 51 + 10, // 10 a 70
        .quantidade = rand() % 100 + 1, // 1 a 100
        .valor = (double)(rand() % 8000 + 1)/100 // 0.00 a 80.00
    };

    return item;
}

// confirma_continuacao: apresenta mensagem se continua ou nao a digitacao
// Input: ENTER para continuar ou EOF para parar
// Output: a mensagem de orientacao (tela) e true/false se for continuar
//  ou nao (valor de retorno da funcao
bool confirma_continuacao(){
    #ifndef __linux__
        #define TERMINO " Ctrl-Z + ENTER "
    #else
        #define TERMINO " Ctrl-D "
    #endif
    printf("Tecle ENTER para inserir um registro\n"
        "ou" TERMINO "para encerrar a digitacao: ");

    // consome o ENTER da ultima leitura
    while(getchar() != '\n')
        /* nada */;
    return (getchar() == '\n');
}


// leitura: realizacao da leitura de uma sequencia de
//  dados de vendas
// Input: os valores para codigo, quantidade e venda de
//  uma sequencia de vendas
// Output: cada registro (struct venda) gravado no arquivo
//  vendas.dat
void leitura(void){
    // abertura do arquivo e verificacao
    FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "wb");
    if(arquivo_vendas == NULL){
        perror(NOME_ARQUIVO);
        exit(1);
```

▷

```
    }
    while(confirma_continuacao()){
        struct venda item = leia_item();
        fwrite(&item, sizeof item, 1, arquivo_vendas);
    }

    // encerramento
    fclose(arquivo_vendas);
}

// aleatorio: gera uma quantidade de registros aleatorios
//  no arquivo de vendas
// Input: a quantidade de registros que serao escritos
// Output: a quantidade especificada de registros com dados
//  aleatorios no arquivo vendas.dat
void aleatorio(void){
    // abertura do arquivo e verificacao
    FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "wb");
    if(arquivo_vendas == NULL){
        perror(NOME_ARQUIVO);
        exit(1);
    }

    printf("Quantidade de registros (0 ou mais): ");
    int quantidade;
    scanf("%d", &quantidade);
    for(int i = 0; i < quantidade; i++){
        struct venda item = item_aleatorio();
        fwrite(&item, sizeof item, 1, arquivo_vendas);
    }

    // encerramento
    fclose(arquivo_vendas);
}
```

Arquivo: codigo/cria_arquivo.c.

No Código 14 está o programa da seção 3, que faz acesso sequencial aos registros arquivados.

Código 14

```
/*
  Este programa apresenta todos os registros contidos no arquivo
  de registro de vendas
  Input: os registros contidos no arquivo vendas.dat
  Output: os campos de cada registro (tela), juntamente com
    mensagens "didaticas" sobre o arquivo e as funcoes
    de manipulacao de arquivos em C

  Jander, 2020

  Cada registro guarda:
    - o codigo do produto (inteiro)
    - a quantidade vendida (inteiro)
    - o valor de venda (double)
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <errno.h>
```

▷

```c
// definicao do registro
struct venda {
    int codigo;
    int quantidade;
    double valor;
};

#define NOME_ARQUIVO "vendas.dat"

// info_posicao: apresenta mensagem contextual sobre a posicao atual
//  do arquivo
// Input: texto com o contexto e o arquivo (FILE*) como parametros
// Output: a mensagem e a posicao corrente do arquivo
void info_posicao(char *mensagem, FILE *stream){
    printf(">>> %s: byte atual = %lu\n", mensagem, ftell(stream));
}

//
// programa principal
//
int main(void){
    // acesso ao arquivo e verificacao de sucesso
    FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "rb");
    if(arquivo_vendas == NULL)
        perror(NOME_ARQUIVO);
    else{
        info_posicao("Arquivo aberto", arquivo_vendas);

        // varredura do arquivo completo
        struct venda item;
        int numero_registro = 0;
        while(fread(&item, sizeof item, 1, arquivo_vendas) > 0){
            printf("Registro %4d: (%d, %d, %.2lf)\n",
                numero_registro++,
                item.codigo, item.quantidade, item.valor);
            info_posicao("Depois da leitura", arquivo_vendas);
        }

        // encerramento
        info_posicao("Antes do fechamento", arquivo_vendas);
        fclose(arquivo_vendas);
    }

    return 0;
}
```

Arquivo: codigo/lista_registros.c.

No Código 15 está o programa da seção 3, que faz acesso sequencial aos registros arquivados.

Código 15

```c
/*
  Este programa apresenta todos os registros contidos no arquivo
  de registro de vendas
  Input: os registros contidos no arquivo vendas.dat
  Output: os campos de cada registro (tela), juntamente com
    mensagens "didaticas" sobre o arquivo e as funcoes
    de manipulacao de arquivos em C
```

▷

```
  Jander, 2020

  Cada registro guarda:
    - o codigo do produto (inteiro)
    - a quantidade vendida (inteiro)
    - o valor de venda (double)
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <errno.h>

// definicao do registro
struct venda {
    int codigo;
    int quantidade;
    double valor;
};

#define NOME_ARQUIVO "vendas.dat"

// info_posicao: apresenta mensagem contextual sobre a posicao atual
//  do arquivo
// Input: texto com o contexto e o arquivo (FILE*) como parametros
// Output: a mensagem e a posicao corrente do arquivo
void info_posicao(char *mensagem, FILE *stream){
    printf(">>> %s: byte atual = %lu\n", mensagem, ftell(stream));
}

//
// programa principal
//
int main(void){
    // acesso ao arquivo e verificacao de sucesso
    FILE *arquivo_vendas = fopen(NOME_ARQUIVO, "rb");
    if(arquivo_vendas == NULL)
        perror(NOME_ARQUIVO);
    else{
        info_posicao("Arquivo aberto", arquivo_vendas);

        // varredura do arquivo completo
        struct venda item;
        int numero_registro = 0;
        while(fread(&item, sizeof item, 1, arquivo_vendas) > 0){
            printf("Registro %4d: (%d, %d, %.2lf)\n",
                numero_registro++,
                item.codigo, item.quantidade, item.valor);
            info_posicao("Depois da leitura", arquivo_vendas);
        }

        // encerramento
        info_posicao("Antes do fechamento", arquivo_vendas);
        fclose(arquivo_vendas);
    }

    return 0;
}
```

Arquivo: codigo/lista_registros.c.