15

Arquivos

Neste capítulo, apresentaremos alguns conceitos básicos sobre arquivos e alguns detalhes da forma de tratamento de arquivos em disco na linguagem C. A finalidade desta apresentação é discutir formas variadas para salvar (e recuperar) informações em arquivos. Com isso, será possível implementar funções para salvar (e recuperar) as informações armazenadas nas estruturas de dados discutidas.

Um arquivo em disco representa um elemento de informação do dispositivo de memória secundária. A memória secundária (disco) difere da memória principal em diversos aspectos. As duas diferenças mais relevantes são: eficiência e persistência. Enquanto o acesso a dados armazenados na memória principal é muito eficiente do ponto de vista de desempenho computacional, o acesso a informações armazenadas em disco é, em geral, extremamente ineficiente. Para contornar essa situação, os sistemas operacionais trabalham com *buffers*, que representam áreas da memória principal usadas como meio de transferência das informações de/para o disco. Normalmente, trechos maiores (alguns kbytes) são lidos e armazenados no *buffer* a cada acesso ao dispositivo. Dessa forma, uma subseqüente leitura de dados do arquivo, por exemplo, possivelmente não precisará acessar o disco, pois o dado requisitado pode já se encontrar no *buffer*. Os detalhes de como esses acessos se realizam dependem das características do dispositivo e do sistema operacional utilizado.

A outra grande diferença entre memória principal e secundária (disco) consiste no fato de as informações em disco serem persistentes, geralmente sendo lidas por programas e pessoas diferentes das que escreveram, o que torna mais prático atribuir nomes aos elementos de informação armazenados no disco (em vez de endereços), falando assim em arquivos e diretórios (pastas). Cada arquivo é identificado por seu nome e pelo diretório em que se encontra armazenado em uma determinada unidade de disco. Os nomes dos arquivos são, em geral, com-

postos pelo nome em si seguido de uma extensão. A extensão pode ser usada para identificar a natureza da informação armazenada no arquivo ou para identificar o programa que gerou (e é capaz de interpretar) o arquivo. Assim, a extensão ".c" é usada para identificar arquivos que têm códigos-fontes da linguagem C, e a extensão ".doc" é, no sistema operacional Windows®, usada para identificar arquivos gerados pelo editor Word da Microsoft®.

Um arquivo pode ser visto de duas maneiras, na maioria dos sistemas operacionais: em "modo texto", como um texto composto de uma seqüência de caracteres, ou em "modo binário", como uma seqüência de bytes (números binários). Podemos optar por salvar (e recuperar) informações em disco em um dos dois modos, texto ou binário. Uma vantagem do arquivo texto é que pode ser lido por uma pessoa e editado com editores de textos convencionais. Em contrapartida, com o uso de um arquivo binário é possível salvar (e recuperar) grandes quantidades de informação de forma mais eficiente. O sistema operacional pode tratar arquivos "texto" de maneira diferente da utilizada para tratar arquivos "binários". Em casos especiais, pode ser interessante tratar arquivos de um tipo como se fossem do outro, desde que tomados os cuidados apropriados.

Para minimizar a dificuldade na manipulação dos arquivos, os sistemas operacionais oferecem um conjunto de serviços para ler e escrever informações em disco. A linguagem C disponibiliza esses serviços para o programador por meio de um conjunto de funções. Os principais serviços que nos interessam são:

- abertura de arquivos: o sistema operacional encontra o arquivo com o nome dado e prepara o buffer na memória;
- leitura do arquivo: o sistema operacional recupera o trecho solicitado do arquivo. Como o buffer contém parte da informação do arquivo, parte ou toda a informação solicitada pode vir dele;
- escrita no arquivo: o sistema operacional acrescenta ou altera o conteúdo do arquivo. A alteração no conteúdo do arquivo é feita inicialmente no buffer para depois ser transferida para o disco;
- fechamento de arquivo: toda a informação contida no buffer é atualizada no disco e a área do buffer utilizada na memória é liberada.

Uma das informações mantidas pelo sistema operacional é um *cursor* que indica a posição de trabalho no arquivo. Para leitura, esse cursor percorre a seqüência de informação existente no arquivo, do início até o fim, conforme os dados vão sendo recuperados (lidos) para a memória. Para escrita, normalmente, os dados são acrescentados quando o cursor se encontra no fim do arquivo.

Nas seções subseqüentes, vamos apresentar as funções mais utilizadas em C para acessar arquivos e discutir diferentes estratégias para tratá-los. Todas as funções da biblioteca padrão de C que manipulam arquivos encontram-se na biblioteca de entrada e saída, com interface em *stdio.h*.

Funções para abrir e fechar arquivos

A função básica para abrir um arquivo é `fopen`:[1]

```
FILE* fopen (char* nome_arquivo, char* modo);
```

`FILE` é um tipo definido pela biblioteca padrão que representa uma abstração do arquivo. Quando abrimos um arquivo, a função tem como valor de retorno um ponteiro para o tipo `FILE`, e todas as operações subseqüentes nesse arquivo receberão esse endereço como parâmetro de entrada. Se o arquivo não puder ser aberto, a função tem como retorno o valor `NULL`.

Devemos passar o nome do arquivo a ser aberto. O nome do arquivo pode ser relativo, e o sistema o procura a partir do diretório corrente (diretório de trabalho do programa), ou pode ser absoluto, e para tanto especificamos o nome completo do arquivo, o que inclui os diretórios, desde o diretório raiz.

Existem diferentes modos de abertura de um arquivo. Podemos abrir um arquivo para leitura ou para escrita e devemos especificar se o arquivo será aberto em modo texto ou em modo binário. O parâmetro `modo` da função `fopen` é uma cadeia de caracteres em que se espera a ocorrência de caracteres que identificam o modo de abertura. Os caracteres interpretados no modo são:

`r`	*read*	Indica modo para leitura;
`w`	*write*	Indica modo para escrita;
`a`	*append*	Indica modo para escrita ao final do existente;
`t`	*text*	Indica modo texto;
`b`	*binary*	Indica modo binário.

Se o arquivo já existe e solicitamos a sua abertura para escrita com modo `w`, o arquivo é destruído e um novo, inicialmente vazio, é criado. Quando solicitamos com modo `a`, o mesmo é preservado, e novos conteúdos podem ser escritos no seu fim. Com ambos os modos, se o arquivo não existe, um novo é criado. Se solicitarmos a abertura de um arquivo para leitura, ele já deve existir; caso contrário a função falha e tem como retorno o valor `NULL`. A função também tem `NULL` como valor de retorno se tentarmos abrir um arquivo para escrita em uma área (diretório) na qual não temos acesso de escrita. Se quisermos abrir um arquivo para simultaneamente ler e escrever, acrescentamos o caractere `+` no modo de abertura. Assim, `r+` indica leitura e escrita em um arquivo já existente e `w+` indica leitura e escrita em um novo arquivo.

Os modos `b` e `t` podem ser combinados com os demais. Mais detalhes podem ser encontrados nos manuais da linguagem C. Em geral, quando abrimos um arquivo, testamos o sucesso da abertura antes de qualquer outra operação, como, por exemplo:

[1] A rigor, os parâmetros do tipo cadeias de caracteres são declarados com o modificador `const`.

```
...
FILE* fp;
fp = fopen("entrada.txt","rt");
if (fp == NULL) {
  printf("Erro na abertura do arquivo!\n");
  exit(1);
}
...
```

Nesse fragmento de código, solicitamos a abertura do arquivo de nome *entrada.txt* para leitura em modo texto. Em seguida, testamos se a abertura do arquivo foi realizada com sucesso.

Após ler/escrever as informações de um arquivo, devemos fechá-lo. Para isso, devemos usar a função `fclose`, a qual espera como parâmetro o ponteiro do arquivo que se deseja fechar. O protótipo da função é:

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

O valor de retorno dessa função é zero, se o arquivo for fechado com sucesso, ou a constante `EOF` (definida pela biblioteca), que indica a ocorrência de um erro.

Arquivos em modo texto

Nesta seção, descreveremos as principais funções para manipular arquivos em modo texto. Também discutiremos algumas estratégias para a organização de dados em arquivos.

Funções para ler dados

A principal função de C para a leitura de dados em arquivos em modo texto é a função `fscanf`, similar à função `scanf` que temos usado para capturar valores inseridos via teclado. No caso da `fscanf`, os dados são capturados de um arquivo previamente aberto para leitura. A cada leitura, os dados correspondentes são transferidos para a memória, e o cursor do arquivo avança, passando a apontar para o próximo dado do arquivo (que pode ser capturado numa leitura subseqüente). O protótipo da função `fscanf` é:

```
int fscanf (FILE* fp, char* formato, ...);
```

Conforme pode ser observado, o primeiro parâmetro deve ser o ponteiro para o arquivo do qual os dados serão lidos. Os demais parâmetros são os já discutidos para a função `scanf`: o formato e a lista de endereços de variáveis que armazenarão os valores lidos. Assim como a função `scanf`, a função `fscanf` também tem como valor de retorno o número de dados lidos com sucesso.

Outra função de leitura muito usada em modo texto é a função `fgetc` que, dado o ponteiro do arquivo, captura o próximo caractere do arquivo (e o cursor avança para o próximo caractere). O protótipo dessa função é:

```
int fgetc (FILE* fp);
```

Apesar de o tipo do valor de retorno ser `int`, o valor retornado é o código do caractere lido. Se o fim do arquivo for alcançado, a constante `EOF` (*end of file*) é retornada.

Outra função muito utilizada para ler linhas de um arquivo é a função `fgets`. Ela recebe como parâmetros três valores: a cadeia de caracteres que armazenará o conteúdo lido do arquivo, o número máximo de caracteres que deve ser lido e o ponteiro do arquivo. O protótipo da função é:

```
char* fgets (char* s, int n, FILE* fp);
```

A função lê do arquivo uma seqüência de caracteres, até que um caractere `'\n'` seja encontrado ou o máximo de caracteres especificado seja alcançado. A especificação de um número máximo de caracteres é importante para evitar invadir memória quando a linha do arquivo for maior do que supúnhamos. Assim, se dimensionarmos nossa cadeia de caracteres, a qual receberá o conteúdo da linha lida, com 121 caracteres, passaremos esse valor para a função, que lerá no máximo 120 caracteres, pois o último será ocupado pelo finalizador de *string* – o caractere `'\0'`. O valor de retorno dessa função é o ponteiro da própria cadeia de caracteres passada como parâmetro ou `NULL` no caso de ocorrer erro de leitura (por exemplo, quando alcançar o final do arquivo).

É importante salientar que a informação lida é sempre a informação apontada pelo cursor do arquivo. Quando abrimos um arquivo para leitura, esse cursor é automaticamente posicionado no início do arquivo. A cada leitura, o cursor avança e passa a apontar para a posição imediatamente após a informação lida. Assim, em uma próxima leitura, captura-se a próxima informação do arquivo.

Funções para escrever dados

Dentre as funções existentes para escrever (salvar) dados em um arquivo texto, vamos considerar as duas mais freqüentemente utilizadas: `fprintf` e `fputc`, análogas, mas para escrita, às funções que vimos para leitura.

A função `fprintf` é similar à função `printf` que temos usado para imprimir dados na saída padrão – em geral, o monitor. A diferença consiste na presença do parâmetro que indica o arquivo para o qual o dado será salvo. O valor de retorno dessa função representa o número de bytes escritos no arquivo. O protótipo da função é dado por:

```
int fprintf(FILE* fp, char* formato, ...);
```

A função fputc escreve um caractere no arquivo. O protótipo é:

```
int fputc (int c, FILE* fp);
```

No primeiro parâmetro, especificamos o código do caractere que queremos escrever (salvar). O valor de retorno dessa função é o próprio caractere escrito, ou EOF se ocorrer um erro na escrita.

Estruturação de dados em arquivos textos

Existem diferentes maneiras de estruturar os dados em arquivos em modo texto, bem como de capturar as informações contidas neles. A forma de estruturar e a forma de tratar as informações dependem da aplicação. A seguir, apresentaremos três modos para representar e acessar dados armazenados em arquivos: caractere a caractere, linha a linha e com palavras-chaves.

Acesso caractere a caractere

Para exemplificar o acesso caractere a caractere, vamos discutir duas aplicações simples. Inicialmente, vamos considerar o desenvolvimento de um programa que conta o número de linhas de um determinado arquivo (para simplificar, vamos supor um arquivo fixo, com o nome "entrada.txt"). Para calcular o número de linhas do arquivo, podemos ler, caractere a caractere, todo o conteúdo do arquivo, e contar o número de ocorrências do caractere que indica mudança de linha, isto é, o número de ocorrências de '\n'.

```
/* Conta número de linhas de um arquivo */

#include <stdio.h>

int main (void)
{
   int c;
   int nlinhas = 0;     /* contador do número de linhas */
   FILE *fp;

   /* abre arquivo para leitura */
   fp = fopen("entrada.txt","rt");
   if (fp==NULL) {
      printf("Não foi possível abrir arquivo.\n");
      return 1;
   }
```