Capítulo 6: Arquivos

Waldemar Celes e Roberto Ierusalimschy

29 de Fevereiro de 2012

1 Funções de entrada e saída em arquivos

Nos capítulos anteriores, desenvolvemos programas que capturam seus dados de entrada via o teclado e exibem seus dados de saída na tela. Quando a quantidade de dados de entrada ou saída é pequena, esta é uma boa estratégia de programação. No entanto, quando a quantidade de dados de entrada ou saída cresce, precisamos de meios mais adequados para especificar os dados de entrada e gerar os dados de saída. É inviável por exemplo, exigir que um usuário forneça, via teclado, um conjunto grande de dados: um erro na digitação de um dado obrigaria a entrada de todos os dados novamente, o que é impraticável. Analogamente, quando um programa exibe uma grande quantidade de dados de saída, é impraticável exibir todos os dados na tela.

A estratégia adequada para trabalhar com grande quantidade de dados é fazer uso de *arquivos de dados*. Se um programa precisa de uma grande quantidade de dados de entrada, é mais adequado que o programa *carregue* (leia) estes dados de um arquivo. Assim, o usuário do programa pode definir os dados de entrada fazendo uso de um editor de texto convencional, criando um *arquivo de entrada*. De forma análoga, se um programa precisa exibir uma grande quantidade de dados de saída, é mais adequado que o programa *salve* (escreva) estes dados em um arquivo, criando um *arquivo de saída*. Assim, após a execução do programa, o usuário pode usar um editor (ou visualizador) de texto convencional para visualizar a saída do programa.

Um arquivo pode ser visto de duas maneiras, na maioria dos sistemas operacionais: em "modo texto", como um texto composto por uma sequência de caracteres, ou em "modo binário", como uma sequência de *bytes* (números binários). Neste curso, vamos trabalhar apenas com arquivos no "modo texto".

Para que possamos ler ou escrever dados em arquivos, é necessário que antes o arquivo seja *aberto*. Cada arquivo é identificado por seu *nome* que, em geral, é composto pelo nome em si seguido de uma extensão. São exemplos de nomes de arquivos: "entrada.txt", "saida.txt", "dados.ent", "dados.sai" etc.

Para *abrir* um arquivo, é necessário definir se o arquivo será aberto para leitura ou para escrita. Para um programa abrir um arquivo para leitura, é necessário que, na execução do programa, o arquivo já exista, pois vamos ler dados deste arquivo. Por outro lado, quando um programa abre um arquivo para escrita, cria-se um novo arquivo onde os dados de saída serão escritos; se já existir um arquivo de mesmo nome, o conteúdo existente é apagado e sobrescrito com os novos dados.

2 Funções da biblioteca padrão

A biblioteca padrão da linguagem C oferece, através da interface *stdio.h*, um conjunto de funções que realizam operações de entrada e saída de dados em arquivos. Nesta seção, vamos apresentar as funções que iremos utilizar no escopo deste curso.

Para abrir um arquivo, a biblioteca padrão oferece a função `fopen`. Esta função serve tanto para abrir um arquivo para leitura como para escrita. A função recebe dois parâmetros, o nome do arquivo que se deseja abrir e o modo de abertura. Para abrir um arquivo para leitura,

podemos especificar o modo `"r"` (*read*); para abrir um arquivo para escrita, podemos especificar o modo `"w"` (*write*). Esta função retorna um *ponteiro* para o tipo `FILE` definido na biblioteca. Um ponteiro é um tipo de variável que serve para armazenar endereços de memória. Neste texto, não vamos discutir o uso de ponteiros em detalhes. Para usar as funções de entrada e saída da biblioteca padrão, basta saber que, na declaração de uma variável do tipo ponteiro, adicionamos um asterisco na frente do nome da variável. Assim, para declarar uma variável de nome `fp` como sendo um ponteiro para o tipo `FILE`, fazemos:

```
FILE *fp;
```

Com a declaração de ponteiro para o tipo `FILE`, podemos invocar a função para abertura de um arquivo. Por exemplo, para abrir o arquivo "saida.txt" para escrita faríamos:

```
fp = fopen("saida.txt", "w");
```

Se fosse para abrir o arquivo com nome "entrada.txt" para leitura, faríamos:

```
fp = fopen("entrada.txt", "r");
```

Se a abertura do arquivo não for bem sucedida, a função retorna o valor definido pela constante simbólica `NULL`. Portanto, podemos detectar e tratar um erro na abertura de um arquivo como no trecho de código a seguir:

```
...
fp = fopen("entrada.txt", "r");
if (fp == NULL) {
   printf("Erro na abertura do arquivo.\n");
   exit(1);   /* aborta programa */
}
...
```

O valor retornado pela função `fopen` deve ser passado como parâmetro para as demais funções para identificar em qual arquivo se deseja fazer a operação de entrada ou saída. Após realizar as operações de entrada e saída no arquivo, este deve ser *fechado* com o uso da função `fclose`.

```
...
fp = fopen("entrada.txt", "r");

/* código de manipulação do arquivo */
...
fclose(fp);
...
```

A principal função de saída em arquivo da biblioteca padrão chama-se `fprintf`, sendo análoga à função `printf` para saída na tela. A diferença é que a função `fprintf` espera um primeiro parâmetro adicional que identifica o arquivo. O trecho de código a seguir mostra de maneira ilustrativa o uso de arquivos para escrever dados:

```
int a;
float b;
FILE *fp = fopen("saida.txt", "w");
...
```

```
fprintf(fp, "Valores: %d %f\n", a, b);
...
fclose(fp);
```

Obviamente, no código anterior, as variáveis a e b devem ter seus valores definidos antes de serem utilizadas pela função `fprintf`.

A principal função de entrada da biblioteca padrão para leitura de dados que se encontram em arquivo chama-se `fscanf`, e é análoga à função `scanf` para captura de dados via teclado, respeitando o formato que lhe é passado como parâmetro. Da mesma forma que a função `fprintf` a diferença está no primeiro parâmetro adicional que identifica o ponteiro para o arquivo. O trecho de código a seguir ilustra a manipulação de um arquivo para leitura de dados:

```
int a;
float b;
FILE *fp = fopen("entrada.txt", "r");
...
fscanf(fp, "%d %f", &a, &b); /* espera ler um valor inteiro e um valor ponto-flutuante */
...
fclose(fp);
```

É importante saber que a função `fscanf` tem como valor de retorno o número de informações que foram lidas com sucesso (isto também vale para a função `scanf`). Este valor de retorno serve para tratarmos erros de leitura ou identificarmos o alcance do final do arquivo, como será explorado em mais detalhes na próxima seção.

3 Exemplos de leitura

Para ilustrar programas que processam dados armazenados num arquivo, vamos considerar a existência de um arquivo que armazena as notas que os alunos obtiveram em uma disciplina. Vamos inicialmente considerar que a primeira linha do arquivo contém um número inteiro que informa a quantidade de notas armazenadas a seguir. Como exemplo ilustrativo, vamos considerar um arquivo com nome "notas.txt" com o seguinte conteúdo:

```
6
7.5
8.4
9.1
4.0
5.7
4.3
```

Podemos escrever um programa que leia as notas existentes no arquivo e exiba, na tela, a média obtida pelos alunos na disciplina. Um código que implementa este programa é mostrado a seguir:

```
#include <stdio.h>

int main (void)
{
  int i;
  int n;
  float nota;
  float soma = 0.0;
  FILE *fp;
```

```
  /* abertura do arquivo para leitura */
  fp = fopen("notas.txt", "r");

  /* teste para verificar se houve algum erro */
  if (fp == NULL) {
    printf("Erro na abertura do arquivo.\n");
    return 1;  /* aborta programa (retorna da função main) */
  }

  /* leitura da quantidade de notas no arquivo */
  fscanf(fp, "%d", &n);

  /* laço para leitura de cada nota */
  for (i=0; i<n; i++) {
    fscanf(fp, "%f", &nota);
    soma = soma + nota;
  }

  /* fechamento do arquivo */
  fclose(fp);

  /* cálculo da média e impressão na tela */
  printf("Media = %.2f\n", soma / n);

  return 0;
}
```

Se este programa for executado para o arquivo ilustrado, será exibido o valor 6.50 que representa a média das notas listadas.

Quando escrevemos programas para processar dados de entrada de um arquivo, procuramos manter o formato dos dados presentes no arquivo o mais flexível possível. O formato do arquivo de notas do exemplo anterior não é muito flexível pois se formos adicionar uma nota no arquivo teremos também que atualizar a primeira linha do arquivo, que representa o número de notas presentes. Para tornar o formato do arquivo mais flexível, podemos pensar em eliminar a informação da primeira linha, isto é, podemos listar no arquivo apenas as notas, e o programa deve ser capaz de exibir a média de todas as notas listadas. Um arquivo de notas com este novo formato é ilustrado por:

```
7.5
8.4
9.1
4.0
5.7
4.3
```

Um programa para exibir a média das notas armazenadas neste arquivo é apresentado a seguir. Para cada nota lida, incrementamos o valor do contador do número de notas, n, necessário para o cálculo da média. Como o arquivo não armazena explicitamente a quantidade de notas listadas, vamos usar o valor de retorno da função `fscanf` para detectar o final do arquivo. Enquanto houver uma linha com uma nota, a função `fscanf` deve retornar o valor 1, uma vez que ela consiga ler 1 valor e atribuí-lo à variável passada como parâmetro. Quando não houver mais notas para serem lidas, a chamada à função `fscanf` deve retornar 0. Sendo assim, colocamos a leitura das notas em um `while` testando como condição para que o laço continue que o valor de retorno de `fscanf` seja igual a 1.

```c
#include <stdio.h>

int main (void)
{
  int n = 0;
  float nota;
  float soma = 0.0;
  FILE *fp;

  /* abertura do arquivo para leitura */
  fp = fopen("notas.txt", "r");

  /* teste para verificar se houve algum erro */
  if (fp == NULL) {
    printf("Erro na abertura do arquivo.\n");
    return 1;  /* aborta programa (retorna da função main) */
  }

  /* leitura de cada nota, até que fscanf não consiga ler uma nova nota */
  while ( fscanf(fp, "%f", &nota) == 1) {
    soma = soma + nota;
    n++;
  }

  /* fechamento do arquivo */
  fclose(fp);

  /* impressão da média na tela */
  printf("Media = %.2f\n", soma / n);

  return 0;
}
```

Como exemplo adicional, podemos agora pensar em um programa que determina a nota máxima encontrada no arquivo. Para determinar o valor máximo de um conjunto de valores, podemos inicializar uma variável com o menor valor possível no conjunto. Como no nosso caso os valores representam notas em uma disciplina, sabemos que o valor mínimo possível é zero. A variável inicializada com o valor mínimo possível representa o valor máximo corrente. No início do programa ela vale o valor mínimo pois ainda não processamos nenhuma nota. Para cada nota encontrada no arquivo, verificamos se seu valor é maior que o valor máximo corrente. Se for, este valor de nota passa a ser o máximo do conjunto. Se fizermos este procedimento para todas as notas, ao final do programa a variável armazenará o valor máximo do conjunto. Um programa que implementa este procedimento é mostrado a seguir:

```c
#include <stdio.h>

int main (void)
{
  float vmax = 0.0;
  float nota;
  FILE *fp;

  fp = fopen("notas.txt","r");
  while (fscanf(fp, "%f", &nota) == 1) {
    if (nota > vmax) {
      vmax = nota;
```

```
    }
  }
  fclose(fp);

  printf("Nota maxima = %f", vmax);

  return 0;
}
```

4 Exemplos de escrita

Nesta seção, vamos considerar um exemplo que escreve em arquivo. Para ilustrar, vamos considerar que numa disciplina os alunos fazem três provas, obtendo três notas que são consideradas para o cálculo da nota final. Considere que as notas obtidas por cada aluno na disciplina estão armazenadas no arquivo "entrada.txt". O formato deste arquivo é ilustrado a seguir: em cada linha do arquivo encontram-se as três notas de cada aluno.

7.5	8.5	7.8
8.4	9.2	6.8
9.1	10.0	9.5
4.0	5.2	4.6
5.7	3.4	4.3
4.3	6.0	5.8

Vamos então desenvolver um programa que processe as notas do arquivo "entrada.txt". Para cada aluno, o programa deve calcular a sua média final ($nf = (p1 + p2 + p3)/3$) e verificar se o aluno foi aprovado ou reprovado. O programa deve gerar um arquivo de saída com o nome "saida.txt". Neste arquivo de saída, para cada linha do arquivo de entrada, deve-se escrever uma linha correspondente com a nota final do aluno e sua situação: "A" se o aluno foi aprovado (isto é, $nf \geq 5.0$) ou "R" se o aluno foi reprovado. O código de uma implementação deste programa é mostrado a seguir:

```
#include <stdio.h>

int main (void)
{
  float p1, p2, p3;
  float nf;
  FILE *entrada, *saida;

  entrada = fopen("entrada.txt","r");
  saida = fopen("saida.txt","w");
  while ( fscanf(entrada, "%f %f %f", &p1, &p2, &p3) == 3 ) {
    nf = (p1 + p2 + p3) / 3;
    fprintf(saida,"%.1f",nf);
    if (nf >= 5.0) {
       fprintf(saida," A\n");
    }
    else {
       fprintf(saida," R\n");
    }
  }
  fclose(saida);
  fclose(entrada);
```

```
  return 0;
}
```

É importante observar que mais de um arquivo pode estar aberto e sendo manipulado ao mesmo tempo em um programa. Se este programa for executado, o usuário pode verificar a criação do arquivo "saida.txt" com as notas finais e situações de aprovação dos alunos, conforme ilustrado a seguir:

7.9 A
8.1 A
9.5 A
4.6 R
4.5 R
5.4 A

Como exemplo adicional, vamos considerar o movimento de um corpo com aceleração constante. Sabemos que, dados a posição inicial, s_0, a velocidade inicial, v_0, e a aceleração, a, a posição e a velocidade do corpo no instante t são dadas, respectivamente, por:

$$\begin{aligned} s &= s_0 + v_0 t + \frac{at^2}{2} \\ v &= v_0 + at \end{aligned}$$

Nosso objetivo é criar um programa que leia do teclado os valores de entrada, s_0, v_0 e a, e salve num arquivo de nome "movimento.txt"os valores das posições e velocidades ao longo do movimento, variando o tempo de 0.0 a 10.0 com incrementos de 1.0. Cada linha do arquivo deve ter os valores: tempo, posição e velocidade, conforme ilustrado a seguir:

0	s_0	v_0
1	s_1	v_1
2	s_2	v_2
3	s_3	v_3
4	s_4	v_4
5	s_5	v_5
6	s_6	v_6
7	s_7	v_7
8	s_8	v_8
9	s_9	v_9
10	s_{10}	v_{10}

Um código que implementa este programa é mostrado a seguir:

```
#include <stdio.h>

int main (void)
{
  float s0, v0, a;
  float t;
  float s, v;
  FILE *saida;

  /* lê valores do teclado */
  printf("Entre com a posicao inicial: ");
```

```
  scanf("%f", &s0);
  printf("Entre com a velocidade inicial: ");
  scanf("%f", &v0);
  printf("Entre com a aceleracao: ");
  scanf("%f", &a);

  /* cria e salva arquivo */
  saida = fopen("movimento.txt","w");
  for (t = 0.0; t <= 10.0; t = t+1.0) {
    s = s0 + v0*t + a*t*t/2.0;
    v = v0 + a*t;
    fprintf(saida,"%f %f %f\n",t,s,v);
  }
  fclose(saida);

  return 0;
}
```

Exercícios

1. Considerando os dois formatos de arquivos com notas obtidas em uma disciplina descritos na Seção 3, escreva programas para:

   (a) Exibir na tela o valor da nota mínima presente no arquivo.

   (b) Exibir na tela o número de alunos aprovados. Considere que um aluno é aprovado se sua nota for maior ou igual a 5.0.

2. Considerando o formato de arquivos com as três notas obtidas por cada aluno em uma disciplina descrito na Seção 4, escreva um programa para processar as notas do arquivo "entrada.txt" e gerar dois novos arquivos: o arquivo "aprovados.txt" com as notas finais dos alunos aprovados e o arquivo "reprovados.txt" com as notas finais dos alunos reprovados.

3. Pede-se:

   (a) Codifique a função $f(x) = 2 + \cos(2\sqrt{x})$. Esta função deve receber o valor de x e retornar o valor de $f(x)$ correspondente, seguindo o seguinte cabeçalho:

   ```
   float f (float x)
   ```

   (b) Usando a função do item acima, escreva um programa (função *main*) que salve em cada linha do arquivo "saida.txt" os valores x_i e $f(x_i)$, com x_i variando de 1 a 100, com incrementos de 1: $x_i = 1, 2, 3, ..., 99, 100$. A figura abaixo ilustra parte do arquivo que deve ser gerado.

   ```
   1    1.58385
   2    1.04863
   3    1.05155
   4    1.34635
   ...
   ```