CAPÍTULO 8

ENTRADA E SAÍDA

Neste capítulo, serão apresentados as operações de C para leitura e gravação em disco. Operações em disco são executadas em arquivos. A biblioteca C oferece um pacote de funções para acessar arquivos de quatro maneiras diferentes:

- Os dados são lidos e escritos um caractere por vez. Oferece as funções getc() e putc().
- Os dados são lidos e escritos como strings. Oferece as funções fgets() e fputs().
- Os dados são lidos e escritos de modo formatado. Oferece as funções fscanf() e fprintf().
- Os dados são lidos e escritos num formato chamado registro ou bloco. É usado para armazenar seqüências de dados como vetores e estruturas. Oferece as funções fread() e fwrite().

Outra maneira de classificar operações de acesso a arquivos é conforme a forma como eles são abertos: em modo texto ou em modo binário. Estes conceitos serão apresentados nas seções a seguir.

8.1 Arquivos Texto

Um arquivo aberto em modo texto é interpretado em C como seqüências de caracteres agrupadas em linhas. As linhas são separadas por um único caractere chamado caractere de nova linha ou LF (*line feed*), de código ASCII 10 decimal. É equivalente ao caractere `'\r'`.

Alguns sistemas operacionais, como o DOS, representam a mudança de linha por dois caracteres: O caractere de retorno de carro ou CR (*carriage return*), de código ASCII 13 decimal (caractere `'\n'`) e o caractere LF. Neste caso, o compilador C converte o par CR/LF em um único caractere de nova linha quando um arquivo em modo texto é lido e converte o caractere de nova linha no par CR/LF quando o arquivo é gravado.

O código em C é independente do sistema operacional que estamos utilizando, tratando a mudança de linha sempre da mesma forma.

O exemplo abaixo lê caracteres de um arquivo texto e armazena em outro arquivo texto. Para executar este exemplo, crie um arquivo texto de nome entrada.txt, contendo algumas frases. Este arquivo pode ser criado em qualquer editor de texto não formatado (evite usar o Word ou o WordPad, por serem de texto formatado. Utilize, por exemplo, o próprio compilador ou o Bloco de Notas). Por exemplo, o arquivo entrada.txt pode ter o seguinte conteúdo:

```
Linguagem C
Programa teste
```

O código do programa é fornecido a seguir.

```
#include <stdio.h>

void main(void)
{
  char ch;
  FILE *in, *out;
```

```
  if((in = fopen("entrada.txt", "rt")) == NULL)
  {
    printf("Impossivel abrir arquivo entrada.txt.");
    return;
  }
  if((out = fopen("saida.txt", "wt")) == NULL)
  {
    printf("Impossivel abrir arquivo saida.txt.");
    return;
  }
  while((ch = getc(in)) != EOF)
    putc(ch, out);
  fclose(in);
  fclose(out);
}
```

Este programa aguarda a entrada de uma linha de texto e termina quando a tecla <ENTER> for pressionada. A linha é gravada no arquivo **saida.txt**.

A estrutura FILE está presente na biblioteca stdio.h e armazena informações sobre o arquivo. Esta estrutura não será discutida neste curso.

8.1.1 As funções fopen() e fclose()

Quando abrimos um arquivo, a informação que recebemos (se o arquivo for aberto) é um ponteiro para a estrutura FILE. Cada arquivo que abrimos terá uma estrutura FILE com um ponteiro para ela.

A função fopen() tem a seguinte sintaxe:

```
FILE* fopen(const char* filename, const char* mode);
```

Esta função recebe como argumentos o nome do arquivo a ser aberto (*filename*) e o modo de abertura (*mode*). Retorna um ponteiro para a estrutura FILE, que armazena informações sobre o arquivo aberto.

O nome do arquivo pode ser fornecido com ou sem o diretório onde ele está localizado. Caso se deseje fornecer o caminho completo do arquivo, lembre-se que a contrabarra em C não é um caractere, e sim um meio de fornecer caracteres especiais. O caractere contrabarra é representado por `'\\'`. Ou seja, em DOS o caminho completo de um arquivo pode ser, por exemplo, `"C:\\temp\\saida.txt"`.

A lista de modos de abertura de arquivos é apresentada a seguir.

"r"	Abrir um arquivo para leitura (*read*). O arquivo deve estar presente no disco.
"w"	Abrir um arquivo para gravação (*write*). Se o arquivo estiver presente, ele será destruído e reinicializado. Se não existir, ele será criado.
"a"	Abrir um arquivo para gravação em anexo (*append*). Os dados serão adicionados ao fim do arquivo existente, ou um novo arquivo será criado.

Além destes modos podem ser adicionados os seguintes modificadores.

"+"	A adição deste símbolo permite acesso de leitura e escrita.
"b"	Abrir um arquivo em modo binário.
"t"	Abrir um arquivo em modo texto.

No exemplo é aberto o arquivo saida.txt, em modo texto, para gravação.

Caso o arquivo possa ser aberto, a função retorna um ponteiro para a estrutura FILE contendo as informações sobre o arquivo. Caso contrário, retorna NULL. No exemplo, é verificado se o arquivo foi aberto com êxito. Caso negativo, o programa apresenta uma mensagem de erro e termina a execução.

O erro na abertura de arquivo pode ser causado por diversos fatores: espaço insuficiente em disco (no caso de gravação), arquivo inexistente (no caso de leitura) etc.

Ao ser aberto um arquivo é necessário que ele seja fechado após utilizado. Isto é feito por meio da função fclose(). Sua sintaxe é a seguinte:

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

Para fechar o arquivo, basta chamar a função fclose() e passar como argumento o ponteiro para a estrutura FILE que contém as informações do arquivo que se deseja fechar. Em caso de êxito, a função retorna 0. Caso contrário, retorna EOF (*end-of-file*). EOF é definido na biblioteca stdio.h através da diretiva **#define** e indica fim de arquivo.

8.1.2 As funções getc () e putc()

A função getc() lê um caractere por vez do arquivo. Sua sintaxe é

```
int getc(FILE* f);
```

A função recebe como argumento um ponteiro para FILE e retorna o caractere lido. Em caso de erro, retorna EOF.

A função putc() é o complemento de getc(). Ela escreve um caractere no arquivo. Sua sintaxe é

```
int putc(int ch, FILE* f);
```

A função recebe como argumentos um caractere e um ponteiro para FILE. O valor retornado é o próprio caractere fornecido. Em caso de erro, a função retorna EOF.

8.1.3 As funções fgets () e fputs()

O exemplo a seguir faz a mesma coisa que o exemplo anterior. A única diferença é que ao invés de acessar os arquivos texto caractere a caractere, eles serão acessados linha a linha.

```
#include <stdio.h>

void main(void)
{
  char ch[1001];
  FILE *in, *out;

  if((in = fopen("entrada.txt", "rt")) == NULL)
  {
    printf("Impossivel abrir arquivo entrada.txt.");
    return;
  }
  if((out = fopen("saida.txt", "wt")) == NULL)
  {
    printf("Impossivel abrir arquivo saida.txt.");
    return;
  }
  while((fgets(ch, 1000, in)) != NULL)
    fputs(ch, out);
```

```
  fclose(in);
  fclose(out);
}
```

A função fgets() tem a seguinte sintaxe.

```
char* gets(char* s, int n, FILE* f);
```

Esta função lê caracteres do arquivo é os coloca na string *s*. A função pára de ler quando lê *n-1* caracteres ou o caractere LF ('\r'), o que vier primeiro. O caractere LF é incluído na string. O caractere nulo é anexado ao final da string para marcar seu final. Em caso de êxito, a função retorna *s*. Em caso de erro ou fim do arquivo, retorna NULL.

A função fputs() é o complemento de fgets(). Ela escreve uma seqüência de caracteres no arquivo. Sua sintaxe é

```
int putc(char* s, FILE* f);
```

A função recebe como argumentos uma string e um ponteiro para FILE. O valor retornado é o último caractere fornecido. Em caso de erro, a função retorna EOF.

Note que a função copia a string fornecida tal como ela é para o arquivo. Se não existir na string, não é inserido o caractere de nova linha (LF) nem o caractere nulo.

8.1.4 As funções fprintf () e fscanf()

Para leitura e gravação de arquivos com formatação, são utilizadas as funções fprintf() e fscanf(). Estas funções são semelhantes a printf() e scanf(), utilizadas ao longo do curso.

O programa a seguir lê dados de um arquivo, executa um processamento com os dados lidos e grava os resultados em outro arquivo, em forma de um relatório de saída.

```
#include <stdio.h>

void main(void)
{
  float distancia, tempo, velocidade;
  FILE *in, *out;

  if((in = fopen("entrada.txt", "rt")) == NULL)
  {
    printf("Impossivel abrir arquivo entrada.txt.");
    return;
  }
  if((out = fopen("saida.txt", "wt")) == NULL)
  {
    printf("Impossivel abrir arquivo saida.txt.");
    return;
  }
  fscanf(in, "%f%f", &distancia, &tempo);
  velocidade = distancia / tempo;
  fprintf(out, "Distancia percorrida: %10.2f km\n", distancia);
  fprintf(out, "Tempo decorrido:      %10.2f h\n", tempo);
  fprintf(out, "Velocidade media:     %10.2f km/h\n", velocidade);
  fclose(in);
  fclose(out);
}
```

Este programa lê do arquivo entrada.txt a distância percorrida e o tempo decorrido. É feito o cálculo da velocidade média e a saída é armazenada num arquivo texto, formatado. As sintaxes de fprintf() e fscanf() são semelhantes às sintaxes de printf() e scanf() exceto pela inclusão do primeiro argumento, que é um ponteiro para a estrutura FILE.

As strings de formatação utilizadas nestas funções são as mesmas utilizadas nas funções printf() e scanf().

8.2 Arquivos Binários

Os dados são armazenados em arquivos binários da mesma forma que são armazenados na memória do computador. Ou seja, ao guardarmos o valor de uma variável **float** num arquivo binário, ela ocupa no arquivo os mesmos 4 bytes que ocupa na memória.

Arquivos guardados em modo binário não são facilmente entendidos por uma pessoa que o esteja lendo. Em compensação, para um programa é muito fácil interpretar este arquivo.

Ao guardarmos valores em modo texto, o compilador converte o valor da variável para uma forma de apresentá-lo através de caracteres. Por exemplo, o número inteiro 20000, apesar de ocupar somente dois bytes na memória, para ser armazenado em um arquivo texto ocupa pelo menos cinco bytes, já que cada caractere ocupa um byte.

Arquivos binários normalmente são menores que arquivos texto, contendo a mesma informação. A velocidade de leitura do arquivo também é consideravelmente reduzida ao utilizarmos arquivos binários.

Em compensação, a visualização do arquivo não traz nenhuma informação ao usuário, pelo fato de serem visualizados os caracteres correspondentes aos bytes armazenados na memória. No caso de variáveis **char**, a visualização é idêntica em arquivos texto e binários.

8.2.1 As funções fread() e fwrite()

O programa abaixo grava um arquivo binário contendo 2 vetores de 20 elemento cada um: o primeiro vetor é de caracteres e o segundo é de variáveis inteiras.

```
#include <stdio.h>

void main(void)
{
  char ch[20] = "Teste geral";
  int valor[20] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
  FILE *out;

  out = fopen("binario.bin", "wb");
  if(out == NULL)
  {
    printf("Impossivel abrir arquivo binario.bin.");
    return;
  }
  fwrite(ch, sizeof(char), 20, out);
  fwrite(valor, sizeof(int), 20, out);
  fclose(out);
}
```

Ao executar este programa, ele gera um arquivo binário, `binario.bin`. Se você tentar visualizar este arquivo num editor como o Bloco de Notas, do Windows, conseguirá distinguir os 20 primeiros bytes, que é o vetor de caracteres. A partir daí, os bytes utilizados pelo compilador para armazenar os valores inteiros não fazem mais sentido quando visualizados.

O programa abaixo lê o arquivo `binario.bin` e apresenta os valores lidos na tela.

```
#include <stdio.h>

void main(void)
{
  char ch[20];
  int valor[20];
  FILE *in;

  in = fopen("binario.bin", "rb");
  if(in == NULL)
  {
    printf("Impossivel abrir arquivo binario.bin.");
    return;
  }
  fread(ch, sizeof(char), 20, in);
  fread(valor, sizeof(int), 20, in);
  for(i = 0; i < 20; i++)
    printf("%c", ch[i]);
  for(i = 0; i < 20; i++)
    printf("\n%d", valor[i]);
  fclose(in);
}
```