Arquivos em C

Na linguagem de programação C, a informação sobre um arquivo é acessada através de um descritor cuja estrutura é definida no arquivo de cabeçalho **`stdio.h`**. Um programa C que vá manipular arquivos deve então incorporar ao início de seu programa fonte a linha de inclusão desse arquivo de cabeçalho:

```
#include <stdio.h>
```

Esse arquivo de cabeçalho define o nome de tipo **`FILE`** associado a essa estrutura. Não é necessário conhecer o formato interno dessa estrutura para manipular arquivos. O programador C, para acessar arquivos, define variáveis ponteiros para este tipo, **`FILE *`**, que são manipuladas diretamente pelas funções da biblioteca padrão de entrada e saída. Tais variáveis são usualmente chamadas de **manipuladores de arquivo**. Assim, a função que vai manipular o arquivo deve incluir a declaração de uma variável desse tipo, como em:

```
FILE *arqFonte;
```

O objetivo de manipular um arquivo é realizar operações de leitura e escrita sobre seu conteúdo. Para que essas operações de transferência de dados tenham sucesso, é preciso que haja a permissão adequada para a operação. Por exemplo, um teclado seria um dispositivo que não aceita saída de dados (escrita), mas apenas entrada (leitura).

Para abrir um arquivo em C, a rotina **`fopen`** é invocada recebendo dois parâmetros. O primeiro é uma *string* com o nome do arquivo que será aberto. O segundo parâmetro é outra *string* que especifica o modo de acesso, que pode conter os caracteres **`r`** (leitura), **`w`** (escrita), **`a`** (escrita ao final -- *append*), e **`b`** (acesso em modo binário). O valor de retorno é o manipulador alocado para o arquivo aberto. O quadro a seguir resume as possibilidades de abertura de arquivo

Modo	Descrição
`r`	Abre um arquivo texto para leitura
`w`	Cria um arquivo texto para escrita
`a`	Adiciona texto ao fim de um arquivo texto
`rb`	Abre um arquivo binário para leitura
`wb`	Abre um arquivo binário para escrita
`ab`	Anexa a um arquivo binário
`r+`	Abre um arquivo texto para leitura/escrita
`w+`	Cria um arquivo texto para leitura/escrita
`a+`	Cria ou anexa a um arquivo texto para leitura/escrita
`r+b`	Abre um arquivo binário para leitura/escrita
`r+b`	Cria um arquivo binário para leitura/escrita
`a+b`	Anexa a um arquivo binário para leitura/escrita

Por exemplo, para realizar a leitura do conteúdo de um arquivo chamado **`teste.asm`**, a seguinte expressão poderia ser usada no programa:

```
arqFonte = fopen("teste.asm", "r");
```

Caso o arquivo não possa ser aberto, a função **fopen** retorna o ponteiro nulo. Assim, para verificar de o arquivo foi aberto sem problemas, é necessário testar o valor de retorno:

```
if (arqFonte != 0) printf("tudo bem");
else printf("ERRO!!!");
```

Encerradas as operações sobre um arquivo, ele deve ser fechado. Isso permite que o sistema operacional libere o espaço ocupado pelas informações sobre o arquivo para que esse mesmo espaço possa ser reocupado para a manipulação de outros arquivos. Esta liberação é importante, uma vez que sistemas operacionais tipicamente limitam a quantidade de arquivos que podem ser abertos simultaneamente devido a restrições de espaço alocado para essas estruturas auxiliares.

Para fechar um arquivo previamente aberto, a rotina fclose pode ser usada. Ela recebe como argumento o manipulador do arquivo e não retorna nenhum valor. Assim, após encerrada a operação com o arquivo a expressão `fclose(arqFonte);` fecha-o.

Existem 2 tipos de arquivos:

Arquivos de texto - onde são gravados apenas os caracteres como se fosse no vídeo e de onde são lidos caracteres como se estivessem sendo digitados no teclado. Por exemplo, os programas fonte (`nomedoprograma.c`) que são digitados, são exemplos de arquivos de texto. Um arquivo texto pode ser visualizado com editor de texto (word, wordpad ou o próprio editor do compilador c).

Arquivos Binários – onde são gravados os dados como estão na memória. Por exemplo, uma variável inteira é gravada com 4 bytes com o conteúdo exato que está na memória. Não é possível ver o conteúdo de um arquivo binário usando um editor de texto. Para isso é necessário ler o arquivo e imprimí-lo no vídeo com **printf**, por exemplo.

Por exemplo, Os dois trechos de arquivo abaixo possuem os mesmo dados :

Arquivo Binário:	`Renato ÌÌÌÌÌ·áõA+ Ì`
Arquivo Texto:	`Renato,32891319,A+`

Arquivos Texto em C

Vídeo, teclado e impressora, podem ser considerados arquivos de texto. Num arquivo de texto em disco, os dados são gravados como se fosse no vídeo ou na impressora. A diferença é que alguns caracteres especiais por exemplo, `\n`, no caso do vídeo ou impressora causam a mudança de linha. No caso de arquivos em disco, são simplesmente colocados no texto gravado.

1 Abrindo Arquivos Texto

Em C, para poder trabalhar em um arquivo, precisamos abri-lo, associando-o a uma variável interna do programa. Para isso, usamos variáveis do tipo **FILE *** (cujo funcionamento interno não nos importa), que podem ser declaradas assim:

```
FILE *entrada;
FILE *saida;
```

O nome das variáveis é, repito, interno ao programa, de modo que poderíamos ter escolhido bacalhau, Dom_Pedro_II ou trifosfato_de_adenosina, independentemente do nome do arquivo com que fôssemos trabalhar.

Mas, calma, a associação entre variável e arquivo ainda não foi feita. Quem faz isso é a função **fopen**, que funciona da seguinte maneira:

```
entrada = fopen("arquivo_entrada.txt", "r");
saida = fopen("arquivo_saida.txt", "w");
```

Essa função precisa de dois parâmetros, dos quais o primeiro é o mais óbvio: o nome do arquivo. O segundo parâmetro diz ao computador o que pretendemos fazer com o arquivo: gravar ("w", de *write*) ou ler dados ("r", de *read*). No final, se tudo tiver dado certo, essas variáveis conterão um tipo de referência aos arquivos que abrimos. São essas variáveis que iremos usar quando formos ler e gravar nossos dados.

Veja que, ao abrir um arquivo para gravação, pode acontecer de já existir um arquivo com o mesmo nome que você pediu. Se isso ocorrer, o arquivo existente será apagado, e o que você gravar ficará no lugar do arquivo antigo. Caso contrário, o programa simplesmente criará um arquivo novo, com o nome que você pediu. Se o que você quer é apenas adicionar dados ao final do arquivo, sem apagar nada, você pode usar, no lugar da letra w, a letra "a" (de *append*).

Problemas

Existem algumas situações que podem impedi-lo de abrir um arquivo:

• O arquivo não existe e você tentou abri-lo para leitura;
• Você não tem permissão para ler ou gravar o arquivo pedido;
• O arquivo já está aberto e/ou bloqueado por outro programa.

Nesses casos, quando você chamar a função **fopen**, a variável referente ao arquivo receberá o valor **NULL** (um valor mais ou menos especial em C, que geralmente indica algum objeto inválido). Para verificar se isso ocorreu, você pode fazer o seguinte teste:

```
if (entrada == NULL)
        printf("Socorro! O arquivo não pôde ser aberto!\n");
```

2 Gravando em Arquivos Texto

Gravar dados em arquivos é realmente fácil, se você estiver acostumado com a função **printf**. A única diferença é que você usa a função **fprintf**, e precisa dizer o arquivo no qual as coisas devem ser escritas. Mas atenção, você não vai dizer o nome do arquivo, mas sim a variável à qual você associou o arquivo agora há pouco!

Veja como se faz isso com alguns exemplos simples:

```
fprintf(saida, "Bom dia!\n");
fprintf(saida, "Resultado: %d\n", resultado);
fprintf(saida, "soma = %10.6f quociente = %10.6f\n", soma, quociente);
```

3 Lendo Arquivos Texto

Para ler arquivos também não há nada de extraordinário: a função **scanf** é trocada pela função **fscanf**, e especificamos (da mesma maneira que com a **fprintf**) o arquivo que queremos usar.

Por exemplo:

```
int n1, n2;
...
fscanf(entrada, "%d %d", &n1, &n2);
```

Cuidados

Quer esteja lendo de um arquivo, quer esteja lendo do teclado, você deve ter cuidado (especialmente no começo de uma linha) ao usar o formato de leitura `"%c"`, usado para ler caracteres. O que ocorre é que o C não faz nenhuma distinção entre os tipos de caracteres quando esse formato é usado.

```
char c;
...
fscanf(entrada, " %c", &c);
```

Similarmente, se você quiser ler dois caracteres separados por espaços, você deve usar o formato `" %c %c"`.

4 Fechando Arquivos Texto

Finalmente, depois que você terminou de se divertir, é necessário fechar os arquivos que você abriu. O fechamento do arquivo serve, basicamente, para liberar a memória que foi utilizada para trabalhar com o arquivo, além de desbloquear o arquivo para uso em outros programas. Essa tarefa é bem simples (e deve ser executada para cada arquivo que foi aberto):

```
fclose(entrada);
fclose(saida);
```

Note que você não deve tentar fechar um arquivo que sequer foi aberto — ou seja, se você constatou que ocorreu um problema (a tal da comparação com NULL).

5 Exemplos

A. Lê e grava um arquivo com números tipo double em seguida mostra o conteúdo do arquivo no vídeo

```
#include <stdio.h>
int main()
{
      FILE *fp;
      char nome_arquivo[32];
      double a;

      // le o nome do arquivo
      printf("\nentre com o nome do arquivo:");
      scanf("%s",nome_arquivo);

      // abre o arquivo para gravação
      fp = fopen(nome_arquivo,"w");

      // le nomes e notas e grava no arquivo
      // le o primeiro numero para começar
```

```
        printf("\n entre com um numero:");
        scanf("%lf", &a);
        while (a != -999)
        {
                fprintf(fp, "%10.4lf", a);
                // le o próximo número
                printf("\nentre com um numero:");
                scanf("%lf", &a);
         }
        fclose(fp);

        // abre o mesmo arquivo para leitura
        fp = fopen(nome_arquivo,"r");
        printf("\n***** conteudo do arquivo criado ******\n");

        // le e mostra todos o conteúdo do arquivo
        while (fscanf(fp, "%lf", &a) != EOF)
                printf("%10.4lf * ", a);

        fclose(fp); // este fclose não é necessário
        return 0;
}
```

Veja o que será impresso, numa execução típica, onde é criado um arquivo de nome testearq.

```
entre com o nome do arquivo:testearq
entre com um numero:1.2345
entre com um numero:-45.435
entre com um numero:34.567
entre com um numero:678.547
entre com um numero:2345.543
entre com um numero:-999

***** conteudo do arquivo criado ******
  1.2345 * -45.4350 * 34.5670 * 678.5470 * 2345.5430 *
```

B. Programa que lê um nome e uma nota, cria um arquivo. Em seguida mostra todo o conteúdo do arquivo. Depois acrescente novos elementos ao arquivo, mostrando novamente o conteúdo do mesmo.

```
#include <stdio.h>

int grava_arquivo (char nome_arq[])
// cria e grava arquivo contendo um nome e uma nota
{
  FILE *fpin;
  char nome[30];
  double nota;

  // abre o arquivo
  fpin = fopen(nome_arq,"w");

  // le nomes e notas e grava no arquivo
  printf("\nnome da pessoa:");
  scanf("%s", nome);
  printf("\nnota desta pessoa:");
  scanf("%lf", &nota);
  while (nota >= 0)
  {
       fprintf(fpin, "%s %4.1lf", nome, nota);
       printf("\nnome da pessoa:");
       scanf("%s",nome);
       printf("\nnota desta pessoa:");
       scanf("%lf", &nota);
```

```
  }
  fclose(fpin);
  return 0;
}

int le_mostra_arquivo (char nome_arq[])
// le arquivo e mostra no vídeo
{
  FILE *fp;
  char nomealuno[30];
  double notaaluno;

  // abre arquivo para leitura
  // abre o arquivo
  fp = fopen(nome_arq,"r");
  // le e mostra no vídeo todas as linhas do arquivo
  printf("\n\n****** conteudo do arquivo *********");
  while (fscanf(fp, "%s%lf", nomealuno, &notaaluno) != EOF)
         printf("\nnome:%s - nota:%5.3lf", nomealuno, notaaluno);
  fclose(fp);
  return 0;
}

int insere_mais(char nome_arquivo[])
{
  FILE *fp;
  char nome[30];
  double nota;

  // abre o arquivo
  fp = fopen(nome_arquivo, "a");

  // le e insere mais elementos no arquivo
  printf("\n\n****** insere mais elementos *********");
  printf("\nnome da pessoa:");
  scanf("%s",nome);
  printf("\nnota desta pessoa:");
  scanf("%lf", &nota);
  while (nota >= 0)
  {
       fprintf(fp, "%s %4.1lf", nome, nota);
       printf("\nnome da pessoa:");
       scanf("%s", nome);
       printf("\nnota desta pessoa:");
       scanf("%lf", &nota);
  }
  fclose(fp);
  return 0;
}

int main()
{
  char nome_arquivo[32];

  // le o nome do arquivo
  printf("\nentre com o nome do arquivo:");
  scanf("%s",nome_arquivo);
  grava_arquivo (nome_arquivo);
  le_mostra_arquivo (nome_arquivo);
  insere_mais(nome_arquivo);
  le_mostra_arquivo (nome_arquivo);
  return 0;
}
```

Quando se digita um string, que é lido por **scanf**, o primeiro caractere branco delimita o string a ser lido. Assim, no exercício acima, os nomes das pessoas digitados não podem ter brancos no meio. Isto é certamente uma limitação grande quando se usa strings com **fprintf** e **fscanf**.

Arquivos Binário em C

As funções que vimos até agora, basicamente lêem e gravam caracteres nos arquivos. Podemos gravar outros tipos de dados nos arquivos, por exemplo **int**, **float**, **double**, ou mesmo uma mistura destes com caracteres.

Por exemplo, considere um arquivo com registros contendo:

```
int cod; // código do produto
char nome[20]; // nome do produto
double preco; // preço do produto
```

Neste exemplo, cada registro lógico do arquivo, contém uma mistura de tipos. Claro que poderíamos ter o mesmo arquivo com o seguinte registro:

```
struct Registro
{
      char cod[12]; // código do produto
      char nome[20]; // nome do produto
      char preco[10]; // preço do produto
};
```

Observe que assim cada registro tem 42 bytes, enquanto que no anterior só 28. Isso ocorre porque números armazenados em binário (como se fosse na memória), ocupam menos espaço que armazenados como cadeias de caracteres. Por isso, a necessidade de arquivos binários.

1 Lendo e Escrevendo Arquivos Binário

Arquivos binários nos permitem ler e escrever estruturas ou mesmo vetores de estruturas de uma vez. Utilizamos as funções, **fread** e **fwrite**. Ambas recebem os mesmos parâmetros:

- **Ponteiro**: Um ponteiro para estrutura (ou vetor) que será lido ou escrito no arquivo. Ou seja, é uma variável que aponta para um *buffer* (a região de memória na qual serão armazenados os dados lidos).
- **Tamanho**: Inteiro que indica o tamanho da estrutura, utilizamos **sizeof** para descobrir o tamanho.
- **Quantidade**: Número de estruturas (maior que 1 quando for vetor).
- **Arquivo**: Ponteiro para o arquivo.

A leitura de arquivos binários é realizada através da função **fread:**

```
k = fread(char *ptr, int tam, int nelem, FILE fp)
```

Esta função lê no vetor apontado por **ptr**, até **nelem** elementos cujo tamanho é **tam**, do arquivo **fp**. A posição do arquivo é avançada do número de caracteres lidos. Se um erro ocorre, o avanço é indeterminado. Retorna o

número de elementos lidos com sucesso, que pode ser menor que **nelem** se chega ao fim do arquivo. Se **tam** ou **nelem** é zero, o valor retornado é zero e nada é lido.

```
k = fwrite(char * ptr, int tam, int nelem, FILE fp)
```

Grava do array apontado por **ptr**, até **nelem** elementos cujo tamanho é **tam**, no arquivo **fp**. A posição corrente do arquivo é avançada do número de caracteres lidos com sucesso. Se um erro ocorre a posição corrente é indeterminada. Retorna o número de elementos lidos com sucesso, que pode ser menor que **nelem** se algum erro ocorre.

Outros exemplos:

```
int val;
fread(&val, sizeof(int),1, leitura);
fwrite(&val, sizeof(int),1, escrita);

int vetor[100];
fread(vetor, sizeof(int), 100, leitura);
fwrite(vetor, sizeof(int), 100, escrita);
```

2 Acessando de Forma Aleatória

Para se fazer procuras e acessos randômicos em arquivos usa-se a função **fseek()**. Esta move a posição corrente de leitura ou escrita no arquivo de um valor especificado, a partir de um ponto especificado. Seu protótipo é:

A sintaxe do comando fseek é:

```
fseek(ponteiro_do_arquivo, posicao_em_bytes, modo_movimentacao);
```

O **ponteiro_do_arquivo** é aquele de quando abrimos o arquivo (Variável FILE *). A **posição_em_bytes,** para onde moveremos o ponteiro interno do arquivo, é um valor **long int**. O modo de movimentação controla como queremos mover o ponteiro. São três modos:

• **SEEK_SET** (constante de valor **0**) - movimenta para a posição indicada (começando a contar do zero, que representa o primeiro byte do arquivo).
• **SEEK_END** (constante de valor **1**)- movimenta para a posição indicada, começando a contar do final do arquivo. Neste caso, o zero representa a posição imediatamente posterior ao último byte do arquivo.
• **SEEK_CUR** (constante de valor **2**)- movimenta a partir da posição atual. Neste caso podemos colocar números negativos, que significa que queremos retroceder com o ponteiro do arquivo. Lembrem-se que qualquer operação de leitura ou gravação em um arquivo move o ponteiro interno deste arquivo, o mesmo números de bytes da leitura (ou gravação).

Quando abrimos um arquivo no modo **append** (a), o ponteiro começa na posição zero, em relação ao final do arquivo. Seria como fazer **fseek** usando **SEEK_END**.

3 Outras funções

A função **rewind()** com assinatura

```
void rewind (FILE *fp);
```

retorna a posição corrente do arquivo para o início.

A função **remove()** com assinatura:

```
int remove (char *nome_do_arquivo);
```

apaga um arquivo especificado.

A função **ftell()** com assinatura:

```
long ftell (FILE *fp)
```

retorna a posição corrente (em bytes) do arquivo, ou –1 em caso de erro.

Com essa função é possível calcular a quantidade de registros do arquivo. Exemplo: arquivo com registro lógico de 50 bytes.

```
// abre o arquivo para leitura
fb = fopen(nomearq, "rb");

// posiciona no final do arquivo
fseek(fb, 0, 2);

// calcula o número de registros do arquivo
k = ftell(fb) / 50;
```

Exemplo: arquivo com o seguinte registro lógico.

```
struct produto {int numprod;
                char nomprod[20];
                int quaprod;
                double preprod;
               };

 // abre o arquivo para leitura
 fb = fopen(nomearq, "rb");

 // posiciona no final do arquivo
 fseek(fb, 0, 2);

 // calcula o número de registros do arquivo
 k = ftell(fb) / sizeof(struct produto);
```

4 Exemplos

C. Escreve e depois lê uma variável float em/de um arquivo binário.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE *pf;
  float pi = 3.1415;
  float pilido;
  if((pf = fopen("arquivo.bin", "wb")) == NULL) // Abre arquivo binário para escrita
  {
      printf("Erro na abertura do arquivo");
      exit(1); // interrompe a execução do programa
  }
```

```
  if(fwrite(&pi, sizeof(float), 1,pf) != 1)     // Escreve a variável pi
      printf("Erro na escrita do arquivo");
  fclose(pf);                                    // Fecha o arquivo

  if((pf = fopen("arquivo.bin", "rb")) == NULL) // Abre o arquivo para leitura
  {
      printf("Erro na abertura do arquivo");
      exit(1); // interrompe a execução do programa
  }

  if(fread(&pilido, sizeof(float), 1,pf) != 1)  // Le em PI lido o valor da variável
                                                // armazenada anteriormente
      printf("Erro na leitura do arquivo");

  printf("\nO valor de PI, lido do arquivo e': %f", pilido);

  fclose(pf);

  return(0); // encerra a execução do programa com sucesso
}
```

D. Cria, lê e mostra arquivo

```
#include <stdio.h>
#include <stdlib.h>

#define maxreg 1000 // define uma constante chamada maxreg com o valor 1000

// cria um arquivo binário formado por 1000 conjuntos de 5 valores inteiros
cria_arq_int(char nome_arq[])
{
  FILE *fp;
  int a[5]; // 5 valores inteiros
  int i, j;

  // abre o arquivo para gravação
  fp = fopen(nome_arq, "wb");
  srand(1234567); // chama função randômica

  for (i = 0; i < maxreg; i++) // define os 1000 conjuntos de 5 inteiros
  {
          a[0] = i;
          for (j = 1; j < 5; j++) a[j] = rand(); // escreve um número randômico
          fwrite(a, sizeof (int), 5, fp);
  }
  fclose(fp); // fecha arquivo
}

// lê o arquivo formado de 1000 conjuntos de 5 inteiros
le_mostra_registro(int nreg, char nome_arq[])
{
  FILE *fp;
  int a[5];
  int i;

  // abre o arquivo para leitura
  fp = fopen(nome_arq, "rb");

  // posiciona no registro nreg
  fseek(fp, nreg * 5 * sizeof(int), 0);

  // verifica se esse registro realmente existe
```

```
  if (fread(a, sizeof (int), 5, fp))
  {
          // mostra no vídeo
          printf("\nregistro %5d - ", nreg);
          for (i = 0; i < 5; i++) printf("%10d", a[i]);
  }
  else printf("\nregistro %5d - inexistente", nreg);

  fclose(fp);
}


main()
{
  char nome[] = "novoarquivo";
  int nr;

  // cria arquivo com max registros
  cria_arq_int(nome);

  while(1)
  {
          // entra com numero do registro a ser lido
          printf("\nregistro a ser lido:");
          scanf("%d", &nr);
          if (nr < 0) break;
          // le e mostra o registro
          le_mostra_registro(nr, nome);
  }
}
```