VOLTAR

# Arquivos

Apresentar os principais conceitos sobre arquivos, dando ênfase nos arquivos texto e binários.

NESTE TÓPICO



## Introdução

Já vimos como podemos receber e enviar dados para usuário por meio do teclado e da tela; agora veremos também como ler e gravar dados em arquivos, o que é aliás muito importante ou até essencial em muitas aplicações.

A principal vantagem na utilização de arquivo está no fato de as informações ficarem armazenadas num meio físico e não somente em memória como visto até agora.

Assim como as funções de entrada/saída padrão (teclado e tela), as funções de entrada/saída em arquivos estão declaradas no cabeçalho *stdio.h*. Aliás, as funções para manipulação de arquivos são muito semelhantes às usadas para entrada/saída padrão. Como já dissemos na seção sobre a entrada e saída padrões, a manipulação de arquivos também se dá por meio de **fluxos** (*streams*).

Um arquivo pode ser visto de duas maneiras, na maioria dos sistemas operacionais: em "modo texto", como um texto composto de uma sequência de caracteres, ou em "modo binário", como uma sequência 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.

## Arquivos em modo texto

Na manipulação de um arquivo, há basicamente três etapas que precisam ser realizadas:

1. abrir o arquivo;
2. ler e/ou gravar os dados desejados;
3. fechar o arquivo.

A manipulação de um arquivo em linguagem c ocorre com a definição do tipo **FILE** que tem como objetivo fazer a comunicação entre a memória principal (RAM) e memória secundária (meios magnéticos), por meio do programa e do sistema operacional.

Em C, todas as operações realizadas com arquivos envolvem seu identificador de fluxo, que é uma variável do tipo "ponteiro de arquivo" (**FILE *)**. Para declarar um identificador de fluxo, faça como se fosse uma variável normal, de acordo com a codificação a seguir.

```
1.  FILE *pont_arq;         // não se esqueça do asterisco! O tipo FILE deve ser escrito
    em maiúsculo.
2.
```

pont_arq será então um ponteiro para um arquivo. É usando este tipo de ponteiro que vamos poder manipular arquivos na linguagem C.

## As Funções Mais Comuns do Sistema de Arquivo

| Função | Operação |
|---|---|
| fopen() | Abre um fluxo |
| fclose() | Fecha um fluxo |
| putc() | Escreve um caractere para um fluxo |
| getc() | Lê um caractere para um fluxo |
| fseek() | Procura por um byte especificado no fluxo |
| fprintf() | É para um fluxo aquilo que printf() é para o console |
| fscanf() | É para um fluxo aquilo que scanf() é para o console |

| | |
|---|---|
| feof() | Retorna verdadeiro se o fim do arquivo é encontrado |
| ferror() | Retorna verdadeiro se ocorreu um erro |
| fread() | Lê um bloco de dados de um fluxo |
| fwrite() | Escreve um bloco de dados para um fluxo |
| rewind() | Reposiciona o localizador de posição de arquivo no começo do arquivo |
| remove() | Apaga um arquivo |

## Abrindo e Fechando um Arquivo

### fopen

Não surpreendentemente, a primeira coisa que se deve fazer para manipular um arquivo é abri-lo. Para isso, usamos a função fopen(), que realiza a abertura de arquivos. Sua sintaxe é:

```
1.  <variável ponteiro> = fopen (char *nome_do_arquivo, char *modo_de_acesso);
```

Abaixo seguem algumas observações sobre a função *fopen*:

- O nome do arquivo determina qual arquivo deverá ser aberto. Este nome deve ser válido no sistema operacional que estiver sendo utilizado. O nome do arquivo deve ser uma string ou com o caminho completo (por exemplo, /usr/share/appname/app.conf ou C:\Documentos\nomes.txt) ou o caminho em relação ao diretório atual (nomes.txt, ../app.conf) do arquivo que se deseja abrir ou criar.

- O modo de acesso é uma string que contém uma sequência de caracteres que dizem se o arquivo será aberto para gravação ou leitura. Depois de aberto o arquivo, você só poderá executar os tipos de ação previstos pelo modo de acesso: não poderá ler de um arquivo que foi aberto somente para escrita, por exemplo. Os modos de acesso estão descritos na tabela a seguir.

| Modo | Significado |
|---|---|
| "r" | Abre um arquivo para leitura. O arquivo deve existir. (O r vem do inglês read, ler) |
| "a" | Acrescenta dados no fim do arquivo existente ("append"). Não apaga o conteúdo pré-existente. (O a vem do inglês append, adicionar, apender) |
| "w" | Cria um arquivo para escrita no início do arquivo. Apagará o conteúdo do arquivo se ele já existir, criará um arquivo novo se não existir. (O w vem do inglês write, escrever) |

Obs: o tipo "w" deve ser utilizado com cuidado, pois caso o arquivo exista, o comando recriará o arquivo, ou seja, você perderá o arquivo criado anteriormente (apaga o arquivo e cria novamente em branco).

O valor de retorno da função fopen() é muito importante! Ele é o identificador do fluxo que você abriu e é só com ele que você conseguirá ler e escrever no arquivo aberto. Como determinado, a função fopen() retorna um ponteiro de arquivo que não deve ter o valor alterado pelo seu programa. Se um erro ocorre quando se está abrindo um arquivo, fopen() retorna um nulo.

exit

Aqui abrimos um parênteses para explicar a função **exit()** cujo protótipo é:

```
1.  void exit (int codigo_de_retorno);
```

Esta função aborta a execução do programa. Pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o *código_de_retorno*. A convenção mais usada é que um programa retorne zero no caso de um término normal e retorne um número não nulo no caso de ter ocorrido um problema.

A função **exit()** se torna importante em casos como alocação dinâmica e abertura de arquivos pois pode ser essencial que uma determinada memória seja alocada ou que um arquivo seja aberto.

Se você deseja abrir um arquivo para escrita com o nome "*teste.txt*" você deve escrever e depois fazer uma validação caso apareça algum erro e o fechamento.

No trecho de código abaixo, é possível observar a abertura do arquivo "*exemplo.txt*" e a utilização da função *exit* no caso do arquivo não existir ou estiver corrompido.

```
1.  FILE *pont_arq;
2.  pont_arq=fopen ("exemplo.txt","w");
3.
4.  if (pont_arq==NULL)
5.  {
6.    printf ("Erro na abertura do arquivo.");
7.    exit 1;
8.  }
```

fclose

Ao terminar de usar um arquivo, você deve fechá-lo. A função *fclose()* é usada para fechar um fluxo que foi aberto por uma chamada à função fopen(). Sua sintaxe é:

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

É importante que se perceba que se deve tomar o maior cuidado para não se "perder" o ponteiro do arquivo. "Perder" neste caso seria se atribuir um valor de outro ponteiro qualquer ao ponteiro de arquivo (perdendo assim o valor original). É utilizando este ponteiro que vamos poder trabalhar com o arquivo.

Se perdermos o ponteiro de um determinado arquivo não poderemos nem fechá-lo. O ponteiro **pont_arq** passado à função **fclose()** determina o arquivo a ser fechado. A função retorna zero no caso de sucesso e você pode usar a função padrão **ferror()** (discutida a seguir) para determinar e reportar quaisquer problemas.

## Exemplo 1

No exemplo a seguir, é possível observar a abertura e fechamento de um arquivo.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
main()
{
   FILE *pont_arq;
   pont_arq = fopen ("texto.txt", "a"); //abertura do arquivo

   if (pont_arq = = NULL)
   {
      printf ("Houve um erro ao abrir o arquivo.\n");
      getch();
      return 1;
   }

   printf ("Arquivo TEXTO criado com sucesso.\n");
   getch();
   fclose (pont_arq); //fechamento do arquivo
   return 0;
}
```

## Exemplo 2

No exemplo a seguir, é possível observar a escrita de uma string em um arquivo utilizando a função *fprintf*.

```c
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
   FILE *pont_arq;
   char frase[80];

   pont_arq = fopen ("texto.txt", "a");
   if (pont_arq == NULL)
   {
      printf ("Houve um erro ao abrir o arquivo.\n");
      getch();
      return 1;
   }
   printf ("\n Escreva uma frase:");
   gets(frase);
   fprintf (pont_arq,"%s \n", frase);
   fclose (pont_arq);
   return 0;
}
```

## Escrevendo em arquivos

Para escrever em arquivos, há quatro funções, das quais três são análogas às usadas para saída padrão: *fputc*, *fputs* e *fprintf*. A tabela a seguir mostra cada função e suas utilidades.

| Saída padrão | Arquivos | Explicação |
|---|---|---|
| putchar | fputc | Imprime apenas um caractere. |
| puts | fputs | Imprime uma string diretamente, sem nenhuma formatação. |
| printf | fprintf | Imprime uma string formatada. |
| N/A | fwrite | Grava dados binários para um arquivo. |

A seguir apresentamos os protótipos dessas funções:

```c
void fputc (int caractere, FILE *fluxo);
void fputs (char *string, FILE *fluxo);
void fprintf (FILE *fluxo, char *formatação, ...);
int fwrite (void *dados, int tamanho_do_elemento, int num_elementos, FILE *fluxo);
```

## Exemplo 3

No exemplo a seguir, é possível observar a escrita de 5 strings em um arquivo utilizando a função *fprintf*.

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
   FILE *pont_arq;
   char frase[80];
   int i;

   pont_arq = fopen ("texto.txt", "a");
   if (pont_arq == NULL)
   {
      printf ("Houve um erro ao abrir o arquivo.\n");
      getch();
      return 1;
   }
   for (i=1; i<=5; i++)
   {
     printf ("\n Escreva uma frase:");
     gets(frase);
     fprintf (pont_arq,"%s \n", frase);
   }

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

## Lendo de arquivos

Novamente, há quatro funções, das quais três se assemelham às usadas para a saída padrão:

- fgetc, fgets: ao chamar a função fgets(), você deve fornecer o ponteiro para a string onde os dados lidos devem ser guardados, além do tamanho máximo dos dados a serem lidos (para que a memória reservada à string não seja ultrapassada).

- fscanf: sintaxe quase igual à de scanf(); só é necessário adicionar o identificador de fluxo no início.

A tabela a seguir mostra cada função e suas utilidades.

| Saída Padrão | Arquivos | Explicação |
|---|---|---|
| getchar | fgetc | Recebe apenas um caractere. |
| gets | fgets | Lê uma string (geralmente uma linha inteira). |
| scanf | fscanf | Recebe uma string formatada. |

| Saída Padrão | Arquivos | Explicação |
|---|---|---|
| N/A | fread | Lê dados binários de um arquivo. |

A seguir apresentamos os protótipos dessas funções:

```
1.  int fgetc (FILE *fluxo)
2.  void fgets (char *string, int tamanho, FILE *fluxo)
3.  void fscanf (FILE *fluxo, char *formatação, ...)
4.  int fread (void *dados, int tamanho_do_elemento, int num_elementos, FILE *fluxo)
```

## Exemplo 4

No exemplo a seguir, é possível observar a recuperação de uma string de 80 caracteres, em um arquivo, utilizando a função *fgets*.

```
1.  #include <conio.h>
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <string.h>
5.  main()
6.  {
7.     FILE *pont_arq;
8.     char frase[80];
9.
10.    pont_arq = fopen ("texto.txt", "r");
11.    if (pont_arq == NULL)
12.    {
13.       printf ("Houve um erro ao abrir o arquivo.\n");
14.       return 1;
15.    }
16.    fgets(frase, 80,pont_arq);
17.    printf ("\n frase recuperada= ");
18.    puts(frase);
19.    getch();
20.    fclose (pont_arq);
21.    return 0;
22. }
```

## Exemplo 5

No exemplo a seguir, é possível observar a recuperação de várias strings, em um arquivo, utilizando a função *fgets*.

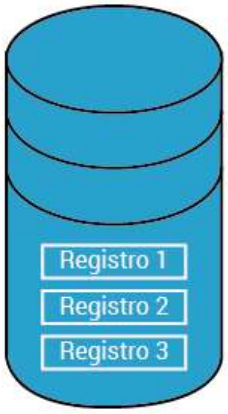
```c
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
   FILE *pont_arq;
   char frase[80];

   pont_arq = fopen ("texto.txt", "r");

   if (pont_arq == NULL)
   {
      printf ("Houve um erro ao abrir o arquivo.\n");
      return 1;
   }

   while (fgets(frase, 80,pont_arq) !=NULL)
   {
     printf ("Frase recuperada= ");
     puts(frase);
   }

   getch();
   fclose (pont_arq);
   return 0;
}
```

## Arquivos em modo binário

O arquivo binário é um conjunto de registros, que por sua vez são constituídos por campos.

Na Figura abaixo é possível observar a estrutura de um arquivo onde estão armazenados todos os dados de clientes de uma Empresa.

Exemplo: O arquivo de Clientes da Empresa, onde estão armazenados os dados de todos os clientes da empresa.




Exemplo do arquivo clientes

As funções para abertura e fechamento de um arquivo binário são as mesmas utilizadas em um arquivo texto (*fopen* e *fclose*). Entretanto, devemos utilizar o caractere *b* para o modo de abertura do arquivo.

No trecho de código abaixo, é possível observar a abertura do arquivo "*pontos.dat*" e a utilização da função *exit* no caso do arquivo não existir ou estiver corrompido.

```
1.  FILE *pont_arq;
2.  pont_arq=fopen ("clientes.dat","wb");
3.
4.  if (pont_arq==NULL)
5.  {
6.    printf ("Erro na abertura do arquivo.");
7.    exit 1;
8.  }
```

A seguir serão apresentadas as funções para escrever e recuperar dados em um arquivo binário.

## Escrevendo e recuperando dados

Para escrever (salvar) dados em arquivos binários, usamos a função *fwrite.* O protótipo dessa função pode ser simplificado por:

```
1.  int fwrite (void* p, int tam, int nelem, FILE* fp);
```

O primeiro parâmetro dessa função representa o endereço de memória cujo conteúdo se deseja salvar em arquivo. O parâmetro *tam* indica o tamanho, em bytes, de cada elemento, e o parâmetro *nelem* indica o número de elementos. Por fim, passa-se o ponteiro do arquivo binário para o qual o conteúdo de memória será copiado.

A função para a leitura (recuperação) dos dados de arquivos binários é análoga, sendo que agora o conteúdo do disco é copiado para o endereço de memória passado como parâmetro. O protótipo da função pode ser dado por:

```
1.  int fread (void* p, int tam, int nelem, FILE* fp);
```

Para exemplificar a utilização dessas funções, vamos considerar que uma aplicação tem um conjunto de pontos armazenados em um vetor. O tipo que define o ponto pode ser:

```
1.  struct ponto{
2.        float x, y, z;
3.  };
4.  typedef struct ponto Ponto;
```

Uma função para salvar o conteúdo de um vetor de pontos pode receber como parâmetros o nome do arquivo, o número de pontos no vetor e o ponteiro para o vetor. Uma possível implementação dessa função é ilustrada a seguir:

```
1.  void salva (char* arquivo, int n, Ponto* vet)
2.  {
3.       FILE* fp = fopen (arquivo, "wb");
4.       if (fp==NULL){
5.           printf ("Erro na abertura do arquivo.");
6.           exit 1;
7.       }
8.       fwrite (vet, sizeof (Ponto), n, fp);
9.       fclose (fp);
10. }
```

A função para recuperar os dados salvos pode ser:

```
1.  void carrega (char* arquivo, int n, Ponto* vet)
2.  {
3.         FILE* fp = fopen (arquivo, "rb");
4.         if (fp==NULL){
5.           printf ("Erro na abertura do arquivo.");
6.           exit 1;
7.         }
8.         fread (vet, sizeof (Ponto), n, fp);
9.         fclose (fp);
10. }
```

Outra grande vantagem oferecida pelo uso de arquivos binários consiste na possibilidade de recuperar apenas parte da informação armazenada. Em um arquivo binário é possível ter o controle de quanto bytes ocupa cada informação armazenada no arquivo. Com isso, podemos alterar a posição do cursor do arquivo, o que permite posicioná-lo para ler uma determinada informação.

A função que permite movimentar o cursor do arquivo tem o seguinte protótipo:

```
1.  int fseek (FILE* fp, long offset, int origem);
```

O primeiro parâmetro indica o arquivo no qual estamos reposicionando o cursor. O segundo parâmetro indica quantos bytes iremos avançar, e o terceiro indica em relação a que posição estamos avançando o cursos: em relação à posição corrente (SEEK_CUR), em relação ao início do arquivo (SEEK_SET) ou em relação ao final do arquivo (SEEK_END).

Para exemplificar, vamos considerar a existência de pontos no espaço 3D salco como exemplificado anteriormente. Vamos, então, escrever uma função que, dado um ponteiro para esse arquivo aberto para leitura, faça a captura do i-ésimo ponto armazenado. Uma possível implementação dessa função é mostrada a seguir.

```
1.  Ponto le_ponto (FILE* fp, int i)
2.  {
3.       Ponto p;
4.       fseek (fp, i*sizeof (Ponto), SEEK_SET);
5.       fread (&p, sizeof (Ponto), 1, fp);
6.       return (p);
7.  }
```

## Quiz

Exercício

Arquivos

INICIAR

## Quiz

Exercício Final

Arquivos

INICIAR

# Referências

MIZRAHI, V. V. *Treinamento em linguagem C*, São Paulo: Pearson, 2008.

SCHILDT, H. C – Completo e Total, São Paulo: Pearson, 2006.

Avalie este tópico

ANTERIOR
Listas encadeadas

Índice

PRÓXIMO
Recursividade

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)