Funções de Manipulação de arquivo

A principio as informações que os programas utilizam são perdidas quando eles são finalizados ou quando o computador é desligado. Isso porque as variáveis de um programa ficam armazenadas na memória primária, que é volátil, isto é, perde seu conteúdo. Quando você não quer perder as informações de seu programa, tendo-as a mão para a sua próxima execução, você deve guardá-las em um arquivo. Os arquivos são estruturas especiais que ficam armazenadas na memória secundária do computador (disquete, disco rígido...) e que servem para guardar as informações enquanto um programa não está em execução, pois elas não são voláteis.

O processo de utilização de um arquivo envolve, no mínimo, três etapas: *criação ou abertura do arquivo*, *gravação ou leitura de dados no arquivo* e *fechamento do arquivo*.

Na primeira etapa, se o arquivo ainda não existir na memória secundária, ele deve ser criado. Caso o arquivo já existir (pelo fato de ter sido criado em uma execução anterior do programa) ele pode ser aberto para que novos dados sejam acrescentados ou para que os dados guardados nele possam ser lidos. Os arquivos podem ser de dois tipos: tipo texto e tipo binário. A escolha se um arquivo é texto ou binário depende do tipo de sua aplicação. Se você for armazenar textos para serem lidos em outro lugar (como um editor de texto ou um relatório), você deve utilizar um arquivo do tipo texto. Se você for armazenar dados sobre alguma pessoa ou objeto (tipo registro ou estrutura), você deve utilizar um arquivo binário, pois, além de ocupar menos espaço no armazenamento das informações, ele as protege um pouco de outros programas bisbilhoteiros.

A segunda etapa é a que efetivamente lê ou grava dados no arquivo. A linguagem C oferece funções específicas de leitura e de escrita de dados em um arquivo. Se o arquivo for do tipo texto (como este que você está lendo), você deve utilizar funções específicas para arquivos-texto. Se ele for binário (do tipo que armazena registros ou estruturas), você deve utilizar as funções de leitura e/ou escrita em arquivos binários.

A terceira etapa consiste em fechar o arquivo, para que seus dados sejam efetivamente gravados e fiquem protegidos até que o arquivo seja aberto novamente.

Vamos agora ver em detalhes cada uma destas etapas e as funções envolvidas nelas.

Etapa I: Abrindo ou criando um arquivo

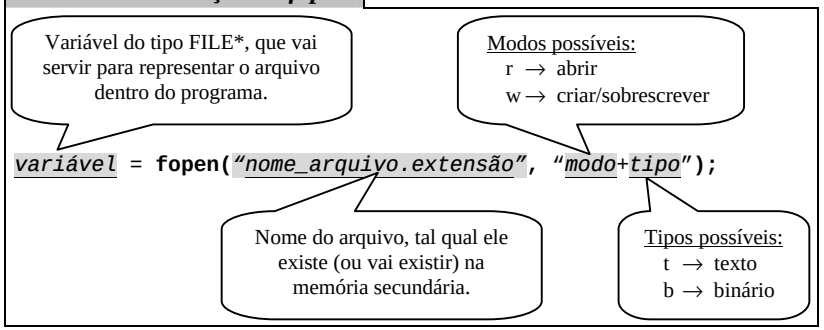
O arquivo é representado na memória secundária por um nome e uma extensão. São exemplos: Apresentação.ppt, autoexec.bat e Trabalho.doc.

Porém, dentro da linguagem C, os arquivos não podem ser manipulados diretamente. Eles precisam representados (referenciados, na verdade) por uma variável do tipo FILE*. Logo, sempre que você for utilizar um arquivo crie uma variável do tipo FILE* para poder utilizá-lo. Todas as funções de manipulação de arquivo necessitam de uma variável deste tipo para poder manipular esse arquivo. Declarar uma variável deste tipo é fácil. Veja:

```
FILE* arquivo; // cria uma variável que manipula arquivos
```

Depois de ter declarado uma variável que vai representar o arquivo, você deve efetivamente tentar abrir ou criar o arquivo. Isso é feito com a função *fopen*. Porém, antes de usar a função *fopen*, você deve decidir se vai abrir ou criar o arquivo (isso vai depender de ele já existir ou não na memória secundária) e também deve decidir se ele vai ser um arquivo do tipo texto ou do tipo binário. Dependendo da sua escolha, a fopen vai ser chamada de uma forma ou de outra, com parâmetros diferentes que indicam as escolhas que você tomou.

Sintaxe de utilização de *fopen*



Logo, para abrir um arquivo o *modo* deve ser "r", e para criar um arquivo (ou sobrescrever um já existente) o *modo* deve ser "w". Tome muito cuidado com o modo "w", pois todos os dados de um arquivo são apagados, caso ele já exista.

Veja os exemplos:

Abrindo um arquivo do tipo texto

```
arquivo = fopen("teste.txt", "r+t");
```

Criando um arquivo do tipo texto

```
arquivo = fopen("teste.txt", "w+t");
```

Note que a única coisa que muda nestes dois exemplos é o *modo*, que no primeiro caso é "r" e no segundo "w".

Se você desejar manipular um arquivo do tipo binário, deve substituir o "t" do tipo por um "b":

Abrindo um arquivo do tipo binário

```
arquivo = fopen("teste.txt", "r+b");
```

Criando um arquivo do tipo binário

```
arquivo = fopen("teste.txt", "w+b");
```

OBS: Antes de seguir em frente, você deve testar se o arquivo foi realmente aberto ou criado. Isso porque as demais funções de manipulação de arquivo não funcionarão se a função *fopen* não tiver funcionado. Para saber se a função *fopen* realmente criou ou abriu o arquivo que você solicitou, teste se a variável que representa o arquivo (a variável *arquivo*, no exemplo) possui algum valor dentro dela. Se ela não possuir um valor (se ele for NULL) é porque o arquivo não foi aberto ou criado:

```
if(arquivo!=NULL)//Só entra aqui se o arquivo foi aberto/criado
{
  // coloque as funções de manipulação de arquivo aqui dentro
}
```


Etapa II: lendo ou gravando dados no arquivo

Depois de ter aberto ou criado um arquivo, e ter uma variável que o represente dentro do programa, você pode começar a gravar ou a ler dados neste arquivo.

Gravando em arquivos-texto

Para gravar, utilize a função *fprintf*, que é muito parecida com a função *printf*. A única diferença é que há um parâmetro (o primeiro) indicando qual é o arquivo onde os dados serão gravados (ou melhor, impressos):

```
int idade = 20;
fprintf(arquivo, "Essa frase vai ser gravada no arquivo\n");
fprintf(arquivo, "%d", idade); // grava a idade no arquivo
fprintf(arquivo, "%s tem %d anos\n", "Leo", idade);
//                 ^^^^^^^^^^^^^ grava "Leo tem 20 anos"
```

Você também pode utilizar *fprintf* para gravar os dados de uma estrutura, mas seus campos têm que ser gravados um-a-um e serão gravados no formato texto:

```
struct tipoEndereco{
  char rua[30];
  int  numero;
} endereco;
     :
fprintf(arquivo, "%s %d\n", endereco.nome, endereco.numero);
```

Lendo a partir de arquivos-texto

Para ler, utilize a função *fscanf*, que é muito parecida com a função *scanf*. A única diferença é que há um parâmetro (o primeiro) indicando qual é o arquivo de onde os dados serão lidos:

```
int  idade;
char nome[20];
fscanf(arquivo, "%d", &idade); // Lê uma idade do arquivo
fscanf(arquivo, "%s %d", nome, &idade);
//               ^^^^^ Lê nome e idade separados por um espaço
```

OBS: Antes de ler ou gravar dados em arquivos binários, é importante se posicionar no local correto. Quando um arquivo é aberto, você fica posicionado na primeira posição do mesmo. Caso você queira fazer uma leitura em outra posição, deve-se posicionar corretamente. Da mesma forma, se você desejar gravar alguma coisa, deve-se posicionar no final do arquivo para que os dados existentes nele não sejam substituídos (veja "posicionando-se em um arquivo binário").

Gravando em arquivos-binários

Para gravar, utilize a função *fwrite*, que grava estruturas inteiras de uma única vez:

```
struct tipoEndereco{
   char rua[30];
   int  numero;
   int  apartamento;
};

tipoEndereco endereço;
char         nome[20];

strcpy(nome, "Laura");
strcpy(endereco.rua, "Lima e Silva");
endereco.numero      = 1047;
endereco.apartamento = 215;

fwrite(&nome, sizeof(nome), 1, arquivo);
//      ^^^^ grava o nome no arquivo
fwrite(&endereco, sizeof(endereco), 1, arquivo);
//      ^^^^^^^^ grava toda a estrutura do tipoEndereco
```

Lendo a partir de arquivos-binários

Para ler, utilize a função *fread*, que é muito semelhante com a função *fwrite:*

```
  // Utiliza as mesmas variáveis do exemplo anterior:

fread(&nome, sizeof(nome), 1, arquivo);
fread(&endereco, sizeof(endereco), 1, arquivo);

printf("%s mora na rua %s, numero %d, apartamento %d.\n",
  nome, endereco.rua, endereco.numero,  endereco.apartamento);
```


Posicionando-se em um arquivo binário

Para se posicionar em um arquivo, utilize a função *fseek.* Aconselha-se que você sempre se posicione no local correto antes de ler ou gravar qualquer dado.

```
// Utiliza as mesmas variáveis do exemplo anterior:

fseek(arquivo, 0, SEEK_END); // posiciona-se no final do arquivo
fwrite(&endereco, sizeof(endereco), 1, arquivo); // grava

fseek(arquivo, 0, SEEK_SET); // posiciona-se no início do
arquivo
fread(&endereco, sizeof(endereco), 1, arquivo); // lê o registro

// posiciona-se no segundo registro:
fseek(arquivo, 2*sizeof(endereço), SEEK_SET);
// obs: sizeof retorna o tamanho do registro!

fread(&endereco, sizeof(endereco), 1, arquivo); // lê o registro
```

OBS: Quando você lê ou grava alguma coisa do arquivo, automaticamente você é colocado na próxima posição do mesmo (próxima linha, em arquivos-texto ou próximo registro, em arquivos-binários).

Etapa III: Fechando um arquivo

Para fechar um arquivo basta chamar a função fclose:

```
fclose(arquivo);
```

Alguns algoritmos de manipulação de arquivos

Inclusão

1) Tentar abrir o arquivo
2) Se o arquivo não existir, criar um novo arquivo
3) Se o arquivo foi aberto ou criado:
    a. Ler os dados a serem gravados no arquivo
    b. Posicionar-se no final do arquivo (ou no primeiro local vago)
    c. Gravar os dados no arquivo
    d. Voltar para (a) enquanto existirem dados a serem gravados
    e. Fechar o arquivo

Consulta

1) Tentar abrir o arquivo
2) Se o arquivo não existir, informar que não há o que consultar e sair
3) Se o arquivo foi aberto:
    a. Posicionar-se no inicio do arquivo
    b. Ler a chave de consulta (o campo a ser consultado)
    c. Ler o registro atual do arquivo em uma variável de memória
    d. Se o registro atual for igual ao requisitado na chave de consulta
        i. Mostrar os dados na tela
    e. Posicionar-se no próximo registro
    f. Se não chegou ao final do arquivo, voltar ao passo (b)
    g. Fechar o arquivo

Exclusão

Semelhante à consulta, porém o passo (d) deve ser modificado da seguinte forma:

    d. Se o registro atual for igual ao requisitado na chave de consulta
        i. Mostrar os dados na tela
        ii. Perguntar se ele realmente deve ser excluído e, caso positivo, posicionar-se na posição onde o registro foi lido e regravá-lo em branco ou utilizar alguma espécie de marca de exclusão

Alteração

Semelhante à consulta, porém o passo (d) deve ser modificado da seguinte forma:

    d. Se o registro atual for igual ao requisitado na chave de consulta
        i. Mostrar os dados na tela
        ii. Solicitar ao usuário que altere os dados desejados
        iii. Confirmar se as alterações estão corretas e, caso positivo, posicionar-se na posição onde o registro foi lido e regravá-lo com as alterações.