

Linguagem C Arquivos

Prof. Edwar Saliba Júnior
Novembro de 2011

Trabalhando com Arquivos

- Um arquivo em Linguagem C pode representar diversas coisas, como:
 - arquivos em disco,
 - uma impressora,
 - um teclado,
 - ou qualquer dispositivos de E/S.

Biblioteca, Tipo e Sintaxe

- A linguagem C dá suporte à utilização de arquivos por meio da biblioteca:

```
stdio.h
```

- Esta biblioteca fornece diversas funções para manipulação de arquivos e também define o tipo:

```
FILE
```

- Uma variável do tipo `FILE` é capaz de identificar um arquivo no disco, direcionando para ele todas as operações. Essa variável é declarada como um ponteiro:

```
FILE *arq, *pont;
```

Tipos de Arquivos

- Na linguagem C, dados podem ser gravados em arquivos do tipo:
 - Binário ou
 - Texto;
- Arquivos do tipo Texto podem ser lidos diretamente;
- Arquivos do tipo Binário devem ser lidos por programas específicos, que convertem os *bits* em informações compreensíveis.

Sistemas Operacionais

- Um arquivo sempre se encontra em um determinado *path* (caminho) no sistema operacional (SO);
- Se este SO for Windows, provavelmente o caminho será algo do tipo:

`C:\Teste\...\NomeDoArquivo.ext`

- Porém se o SO for Unix, GNU/Linux, Mac ou assemelhados, então provavelmente o caminho será algo do tipo:

`/home/.../NomeDoArquivo.ext`

- **Observação:** A maioria dos exemplos aqui trabalhados foram feitos utilizando-se o SO GNU/Linux.

Abrindo Arquivos

- A função `fopen ()` abre um arquivo, retornando o ponteiro associado a este arquivo.
- A sintaxe correta para utilização é:

```
FILE *p;
```

```
p = fopen(nome_arquivo, modo_abertura);
```

- Onde:
 - `nome_arquivo` representa o nome do arquivo que se deseja abrir, podendo conter o caminho onde o arquivo se encontra;
 - `modo_abertura` representa como o arquivo será aberto. A tabela a seguir apresenta todos os modos de abertura.

Modos de Abertura

Comando	O que o comando faz e (Operações Realizadas no Arquivo)
r	Abre um arquivo de texto (leitura).
w	Cria um arquivo de texto (escrita).
a	Anexa novos dados a um arquivo de texto.
rb	Abre um arquivo binário (leitura).
wb	Cria um arquivo binário (escrita).
ab	Anexa novos dados a um arquivo binário.
r+	Abre um arquivo de texto (leitura e escrita).
w+	Cria um arquivo de texto (leitura e escrita).
a+	Anexa novos dados ou cria um arquivo texto (leitura e escrita).
rb+	Abre um arquivo binário (leitura e escrita).
wb+	Cria um arquivo binário (leitura e escrita).
ab+	Anexa novos dados a um arquivo binário (leitura e escrita).

Observação 01

- Quando a função `fopen ()` é utilizada para abrir um arquivo no modo escrita (`w` e `wb`), duas situações podem ocorrer:
 - Se o arquivo NÃO existir, ele será criado;
 - Se o arquivo JÁ existir, ele será sobreposto por um novo arquivo vazio.

Observação 02

- Ao tentar abrir um arquivo, se a função `fopen ()` for executada sem problemas, a variável `arq` (ponteiro para `FILE`) receberá o endereço de memória ocupado pelo arquivo;
- Caso ocorra algum erro na abertura do arquivo a variável `arq` receberá o valor `NULL`;
- Sendo assim, é recomendada a utilização de um teste na abertura de um arquivo.

Exemplo

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *arq;
6
7      arq = fopen("/media/DataDisk/Temp/Exemplos/Teste.txt","r");
8
9      if(arq == NULL)
10         printf("\nErro ao tentar abrir arquivo.");
11     else{
12         printf("\nSucesso na abertura do arquivo.");
13     }
14
15     // Continuação do programa.
16
17     return 0;
18 }
```

Atenção!

- A função `fopen ()` não é capaz de criar diretórios (pastas);
- Para criar um diretório use a função `mkdir ()`. Sua sintaxe é:

```
mkdir( "c: \\exemplo\\teste" );
```

- Quando formos referenciar um caminho de um arquivo (para qualquer tipo de operação) em Windows, é necessário que coloquemos sempre “\\”, pois, senão o compilador da linguagem poderá confundir a “\” com o caractere de controle usado em “\t” ou “\n”.

Fechando um Arquivo

- A função `fclose ()` fecha um arquivo;
- Quando ocorrer algum erro durante a execução do programa, poderá haver perda de dados ou até mesmo perda do arquivo;
- Por isto é sempre recomendado que ao acabar de utilizar um arquivo ele seja fechado;
- Sintaxe da função:

```
fclose ( arq ) ;
```
- Onde:
 - `arq` é a referência para um arquivo aberto.

Exemplo

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *arq;
6
7      arq = fopen("/media/DataDisk/Temp/Exemplos/Teste.txt","r");
8
9      if(arq == NULL)
10         printf("\nErro ao tentar abrir arquivo.");
11     else{
12         printf("\nSucesso na abertura do arquivo.");
13     }
14
15     fclose(arq);
16
17     return 0;
18 }
```

Observação

- Quando a função `fclose()` é executada, gera como resultado um número inteiro;
- Se este número for igual a zero, significa que o arquivo foi fechado corretamente. Caso contrário ocorreu um erro na operação;
- Erros são passíveis de serem capturados.

Capturando Erros Durante o Uso de Arquivos

- A função `error ()` detecta se ocorreu algum erro durante uma operação com arquivos;
- Sua sintaxe é:

```
error(FILE *arq);
```
- Esta função retorna um número inteiro e deve ser chamada logo depois que qualquer outra função for invocada. Se o número retornado for diferente de zero, significa que ocorreu um erro durante a última operação realizada com o arquivo.

Gravando Caracteres em um Arquivo

- A função `fputc ()` escreve um caractere em um arquivo.
- Sintaxe:

```
fputc(char ch, FILE *arq);
```
- Onde:
 - `ch` é o caractere que será escrito no arquivo;
 - `arq` é a referência para o arquivo onde o caractere será escrito.

Exemplo

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *arq;
6      char caractere;
7
8      arq = fopen("/media/DataDisk/Temp/Exemplos/Caracteres.txt","w");
9
10     if(arq == NULL)
11         printf("\nErro ao tentar abrir arquivo.");
12     else{
13         do{
14             printf("\nDigite um caractere qualquer ou 'f' para terminar: ");
15             scanf(" %c",&caractere);
16
17             fputc(caractere,arq);
18
19             if(ferror(arq))
20                 printf("Erro na gravação do caractere: %c", caractere);
21         }while(caractere != 'f');
22     }
23
24     fclose(arq);
25
26     return 0;
27 }
```

Lendo Caracteres de um Arquivo

- A função `fgetc ()` lê um caractere de um arquivo.
- Sintaxe:

```
int fgetc(FILE *arq);
```

- Onde:
 - `arq` é a referência para o arquivo de onde será lido o caractere.
- Se a execução for bem sucedida, gerará como retorno o valor do caractere lido (tabela ASCII). Caso contrário o valor devolvido será EOF.

Exemplo

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *arq;
6      char caractere;
7
8      arq = fopen("/media/DataDisk/Temp/Exemplos/Caracteres.txt", "r");
9
10     if(arq == NULL)
11         printf("\nErro ao tentar abrir arquivo.");
12     else{
13         while(!feof(arq)){
14             caractere = fgetc(arq);
15
16             if(ferror(arq))
17                 printf("\nErro de leitura de arquivo.");
18             else
19                 printf("\n0 caractere lido foi: %c", caractere);
20         }
21     }
22
23     fclose(arq);
24
25     return 0;
26 }
```

Gravando Sequência de Caracteres num Arquivo

- A função `fputs ()` escreve uma sequência de caracteres em um arquivo.

- Sintaxe:

```
fputs(char *sequencia, FILE *arq);
```

- Onde:

- `sequencia` armazena a sequência de caracteres que será escrita no arquivo;
- `arq` é a referência para o arquivo em que será gravada a sequência.

Exemplo

```
1  #include <stdio.h>
2  #define TAM 30
3
4  int main()
5  {
6      FILE *arq;
7      char sequencia[TAM];
8
9      arq = fopen("/media/DataDisk/Temp/Exemplos/Teste.txt","a");
10
11     if(arq == NULL)
12         printf("\nErro ao tentar abrir arquivo.");
13     else{
14         do{
15             printf("\nDigite uma sequência de caracteres ou 'fim' para terminar: ");
16             gets(sequencia);
17
18             fputs(sequencia,arq);
19
20             if(ferror(arq))
21                 printf("\nErro na leitura da sequência de caracteres.");
22             else
23                 printf("\nGravação efetuada com sucesso.");
24         }while(strcmp(sequencia,"fim") != 0);
25     }
26
27     fclose(arq);
28
29     return 0;
30 }
```

Lendo Sequências de Caracteres de um Arquivo

- A função `fgets ()` lê uma cadeia de caracteres armazenadas em um arquivo;
- A cadeia de será formada por todos os caracteres existentes, da posição atual do ponteiro do arquivo até uma marca de nova linha (`\n`) ou até que `tam - 1` caracteres sejam lidos (`tam` é um dos parâmetros utilizados pela função);

- Sintaxe:

```
fgets(char *cadeia, int tam, FILE *arq);
```

- Onde:
 - `cadeia` armazena a cadeia de caracteres obtida do arquivo;
 - `tam` indica que a quantidade máxima de caracteres lidos será `tam - 1`;
 - `arq` é a referência para o arquivo.

Exemplo

```
1  #include <stdio.h>
2  #define TAM 30
3
4  int main()
5  {
6      FILE *arq;
7      char sequencia[TAM];
8
9      arq = fopen("/media/DataDisk/Temp/Exemplos/Teste.txt","r");
10
11     if(arq == NULL)
12         printf("\nErro ao tentar abrir arquivo.");
13     else{
14         while(!feof(arq)){
15             fgets(sequencia,TAM,arq);
16
17             if(ferror(arq))
18                 printf("Erro na leitura da sequência de caracteres.");
19             else
20                 printf("\nSequência lida: %s",sequencia);
21         }
22     }
23
24     fclose(arq);
25
26     return 0;
27 }
```

Gravando Qualquer Tipo de Dado

- Arquivos em linguagem C não podem ser associados a um tipo primitivo de dados ou a um registro (*struct*);
- Arquivos armazenam uma sequência de caracteres ou *bytes*;
- Entretanto, às vezes temos que ler parte do conteúdo de um arquivo e gravar diretamente em uma variável `int` ou `float`, ou ainda em um registro (*struct*). Também é importante conseguir pegar o conteúdo de variáveis e gravá-lo diretamente em um arquivo;
- Quando isto for necessário, o programa deverá trabalhar com arquivos **Binários**.

Arquivos Binários

- Toda vez que uma operação de leitura ou escrita for realizada, deverá ser informado o número de *bytes* que serão lidos ou gravados;
- Para isto, a função `sizeof ()` será utilizada intensamente, uma vez que por meio dela é possível descobrir quantos *bytes* uma variável (de qualquer tipo, incluindo *struct*) ocupa.

Gravação em Arquivos Binários

- A função `fwrite()` pode escrever qualquer tipo de dado e não apenas caracteres ou sequências de caracteres;

- Sintaxe:

```
fwrite(void *mem, size_t qtd_bytes, size_t cont, FILE *arq);
```

- Onde:

- `mem` representa a variável que armazena o conteúdo a ser gravado no arquivo;
- `qtd_bytes` representa o total em *bytes* que será gravado;
- `cont` representa o número de blocos de tamanho `qtd_bytes` que será gravado;
- `arq` é a referência para o arquivo onde as informações serão gravadas;
- Quando a função `fwrite()` for bem sucedida, gerará como retorno um valor igual ao número de gravações realizadas (igual ao parâmetro `cont` informado);
- Se ocorrer algum erro, o valor retornado será menor que `cont`.

Exemplo em (.odt) ou (.pdf)

Leitura em Arquivos Binários

- A função `fread()` pode ler qualquer tipo de dado e não apenas caracteres ou sequências de caracteres;

- Sintaxe:

```
fread(void *mem, size_t qtd_bytes, size_t cont, FILE *arq);
```

- Onde:

- `mem` representa a variável que armazena o conteúdo a ser gravado no arquivo;
- `qtd_bytes` representa o total em *bytes* que será gravado;
- `cont` representa o número de blocos de tamanho `qtd_bytes` que será gravado;
- `arq` é a referência para o arquivo onde as informações serão gravadas;
- Quando a função `fread()` for bem sucedida, gerará como retorno um valor igual ao número de gravações realizadas (igual ao parâmetro `cont` informado);
- Se ocorrer algum erro, o valor retornado será menor que `cont`.

Exemplo em (.odt) ou (.pdf)

Encontrando o Final do Arquivo

- A função `feof ()` descobre se o final do arquivo foi encontrado.
- Ela retorna um número inteiro. Quando este número for zero, significa que o fim do arquivo ainda não foi atingido. Qualquer outro valor significa que o fim do arquivo foi encontrado;
- Sintaxe:
`feof (arq) ;`
- Onde:
 - `arq` é a referência para o arquivo a ser analisado.

Voltando o Cursor ao Início do Arquivo

- Cursor é um ponteiro que indica a partir de que posição, dentro de um arquivo, uma operação será realizada;
- Exemplo:
 - Quando um arquivo acaba de ser aberto, seu cursor está apontando para a posição zero, ou seja, onde está o primeiro *byte* do arquivo;
 - Caso seja feita um leitura como o comando `fread()`, o cursor se movimentará a quantidade de *bytes* lidos;
- A função `rewind()` posiciona o cursor de volta ao início do arquivo;
- Sintaxe:

```
rewind(FILE *arq);
```

Reposicionando o Cursor de um Arquivo

- A função `fseek()` é utilizada especialmente para mudar a posição do cursor sem que haja necessidade de leituras ou escritas no arquivo;
- Sintaxe:

```
fseek(FILE *arq, long qtd_bytes, int posicao);
```
- Onde:
 - `arq` representa o arquivo que será percorrido pela função `fseek()`;
 - `qtd_bytes` representa a quantidade de bytes que o cursor será movimentado a partir da `posicao`;

Continua...

Reposicionando o Cursor de um Arquivo

- `posicao` ponto a partir do qual a movimentação será executada, podendo assumir três valores:
 - `SEEK_SET` – permite a movimentação de `qtd_bytes` a partir da posição inicial do arquivo;
 - `SEEK_CUR` – permite a movimentação de `qtd_bytes` a partir da posição corrente do arquivo;
 - `SEEK_END` – permite a movimentação de `qtd_bytes` a partir do fim do arquivo;
- Exemplos:
 - ```
fseek(cli, sizeof(Cliente) * 2, SEEK_SET);
```
  - Movimenta uma quantidade de *bytes* referente a duas vezes o tamanho de um registro do tipo *Cliente* (*struct*), a partir do início do arquivo em direção ao fim;

Continua...

## Reposicionando o Cursor de um Arquivo

```
fseek(cli, sizeof(Cliente) * cont, SEEK_CUR);
```

- Movimenta uma quantidade de *bytes* referente a *cont* vezes o tamanho de um registro do tipo *Cliente (struct)*, a partir da posição corrente do cursor em direção ao fim do arquivo;

```
fseek(cli, sizeof(Cliente), SEEK_END);
```

- Movimenta uma quantidade de *bytes* referente ao tamanho de um registro do tipo *Cliente (struct)*, a partir do fim do arquivo em direção ao início do arquivo.

## Apagando um Arquivo

- A função `remove ( )` apaga um arquivo;

- Sintaxe:

```
remove(char *nome_arq);
```

- Onde:

- `nome_arq` indica o nome físico do arquivo que será removido, podendo ser incluído o caminho;

- Quando executada com êxito, esta função retorna zero. Caso contrário devolverá um valor diferente de zero;

- Exemplo:

```
remove("c:\exemplo\teste\cliente.dat");
```

## Renomeando um Arquivo

- A função `rename ( )` apaga um arquivo;

- Sintaxe:

```
rename(char *nome_atual, char *nome_novo);
```

- Onde:

- `nome_atual` indica o nome físico atual do arquivo que será renomeado, pode-se incluir o caminho;
- `nome_novo` indica o novo nome físico que se pretende dar ao arquivo que será renomeado, pode-se incluir o caminho;

- Exemplo:

Continua...

# Renomeando um Arquivo

- Exemplos:

```
rename("c:\\teste\\clientes.dat",
"c:\\teste\\dados.dat");
```

- No exemplo acima o arquivo “clientes.dat” tem seu nome físico trocado para “dados.dat”;

```
rename("c:\\teste\\clientes.dat",
"c:\\dados.dat");
```

- No exemplo acima o arquivo “clientes.dat” que se encontra no diretório “teste”, será removido de lá e copiado para o diretório raiz com o nome de “dados.dat”.

## Bibliografia

- ASCENCIO, Ana F. G.; CAMPOS, Edilene A. V.  
**Fundamentos da Programação de computadores.** 2 ed.  
São Paulo: Pearson, 2007.