

# Programação de Computadores

Profs. Marilton Sanchotene de Aguiar e Giovani Parente Farias

{marilton, giovani.pfarias}@inf.ufpel.edu.br

---

Cursos de Ciência e Engenharia de Computação  
Centro de Desenvolvimento Tecnológico  
Universidade Federal de Pelotas





## Semana VIII

### Arquivos



- Arquivos vs. *Streams*
  - O sistema de E/S da linguagem C fornece uma interface consistente ao programador, independentemente do dispositivo real que é acessado.
  - Este sistema (que é uma abstração) é chamado de *stream* e o dispositivo real é chamado de arquivo.
  - Daí decorre que, em C, todos os dispositivos são encarados como arquivos



- O sistema de arquivos de C é projetado para trabalhar com uma ampla variedade de dispositivos, incluindo: terminais, acionadores de disco, e acionadores de fita.
- Embora cada um dos dispositivos seja muito diferente, o sistema de arquivo com *buffer* transforma-os em um dispositivo lógico chamado de *stream*.
- Todas as *streams* comportam-se de forma semelhante.
- Pelo fato de as *streams* serem totalmente independentes do dispositivo, a mesma função pode escrever em um arquivo em disco ou em algum outro dispositivo, como o console.
- Existem dois tipos de *streams*: texto e binária.



- Um *stream* de texto é uma sequência de caracteres.
- O padrão C ANSI permite (mas não exige) que uma *stream* de texto seja organizada em linhas terminadas por um caractere de nova linha.
- Porém, o caractere de nova linha é opcional na última linha e é determinado pela implementação.



- Uma *stream* binária é uma sequência de bytes com uma correspondência de um para um com aqueles encontrados no dispositivo externo.
- O número de bytes escritos (ou lidos) é o mesmo que o encontrado no dispositivo externo.
- Porém, um número definido pela implementação de bytes nulos pode ser acrescentado a um *stream* binário.
- Esse bytes nulos poderiam ser usados para aumentar a informação para que ela preenchesse um setor de um disco, por exemplo.



- Em C, um arquivo pode ser qualquer coisa, desde um arquivo em disco até um terminal ou uma impressora.
- Associa-se um *stream* com um arquivo específico realizando uma operação de abertura.
- Uma vez o arquivo aberto, informações podem ser trocadas entre ele e o seu programa.
- Nem todos os arquivos apresentam os mesmos recursos.
- Por exemplo, um arquivo em disco pode suportar acesso aleatório (sequencial), enquanto um teclado não pode.
- Isso releva um ponto importante sobre o sistema de E/S de C: todas as *streams* são iguais, mas não todos os arquivos.



- Se o arquivo pode suportar acesso aleatório
  - Abrir este arquivo inicializa o indicador de posição no arquivo para o começo do arquivo
  - Quando cada caractere é lido ou escrito no arquivo, o indicador de posição é incrementado, garantindo progressão através do arquivo.
- Um arquivo é desassociado de uma *stream* específica por meio de uma operação de fechamento.
- Se um arquivo aberto para saída for fechado, o conteúdo, da sua *stream* associada será escrito no dispositivo externo. Este processo é geralmente referido como descarga (*flushing*) da *stream* e garante que nenhuma informação seja acidentalmente deixada no *buffer* do disco.





- Todos os arquivos são fechados automaticamente quando o programa termina normalmente, com
  - `main()` retornando ao sistema operacional ou
  - uma chamada a `exit()`
- Os arquivos não são fechados quando um programa quebra (*crash*) ou quando ele chama `abort()`.
- Cada *stream* associada a um arquivo tem uma estrutura de controle de arquivo do tipo `FILE`. Esta estrutura é definida no cabeçalho `stdio.h`.
- O sistema de arquivos C ANSI é composto de diversas funções interrelacionadas. Estas funções exigem o cabeçalho `stdio.h` e a maioria começa com a letra “f”.



| Nome      | Função   |
|-----------|--|
| fopen()   | abre um arquivo  |
| fclose()  | fecha um arquivo   |
| puts()    | escreve um caractere                                     |
| fputs()   | escreve um caractere em um <i>stream</i>                 |
| gets()    | lê um caractere  |
| fgets()   | lê um caractere de um <i>stream</i>                      |
| fseek()   | posiciona um arquivo em um byte específico               |
| fprintf() | o mesmo que printf() para console                        |
| fscanf()  | o mesmo que scanf() para console                         |
| feof()    | devolve verdadeiro para fim de arquivo                   |
| ferror()  | devolve verdadeiro se ocorreu algum erro                 |
| rewind()  | recoloca o indicador de posição para o início do arquivo |
| remove()  | apaga um arquivo   |
| fflush()  | descarrega um arquivo                                    |

- Um ponteiro de arquivo é um ponteiro para informações que definem várias coisas sobre o arquivo: nome, status e a posição atual do arquivo
- Um ponteiro de arquivo é uma variável ponteiro do tipo `FILE`.
- Para ler ou escrever arquivos, seu programa precisa usar ponteiros de arquivo. Para obter uma variável ponteiro de arquivo, use o comando:

```
FILE *fp;
```




- A função `fopen()` abre uma stream para uso e associa um arquivo a ela
- Ela retorna o ponteiro de arquivo associado a este arquivo

```
FILE fopen(const char* nomearq, const char* modo);
```

onde `nomearq` é um ponteiro para uma cadeia de caracteres que forma um nome válido de arquivo e pode incluir uma especificação de caminho de pesquisa (*path*).





| <b>Modo</b> | <b>Significado</b>                            |
|-------------|---|
| r           | abre arquivo texto para leitura               |
| w           | cria arquivo texto para escrita               |
| a           | anexa ao arquivo texto                        |
| rb          | abre arquivo binário para leitura             |
| wb          | cria arquivo binário para escrita             |
| ab          | anexa ao arquivo binário                      |
| r+          | abre arquivo texto para leitura/escrita       |
| w+          | cria arquivo texto para leitura/escrita       |
| a+          | anexa ao arquivo texto para leitura/escrita   |
| r+b         | abre arquivo binário para leitura/escrita     |
| w+b         | cria arquivo binário para leitura/escrita     |
| a+b         | anexa ao arquivo binário para leitura/escrita |

## Exemplo

```
FILE *fp;  
if((fp = fopen( "arquivo.txt", "w")) == NULL ) {  
    printf("nao foi possivel criar o arquivo\n");  
    exit(1);  
}
```

Neste caso, qualquer erro na abertura do arquivo será detectado (e.g., disco cheio ou protegido contra gravação).



- Confirmar o sucesso de `fopen()` antes de tentar qualquer outra operação sobre o arquivo!
- Se você usar `fopen()` para abrir um arquivo com permissão para escrita, qualquer arquivo já existente com esse nome será apagado e um novo arquivo será iniciado
- Se nenhum arquivo com este nome existe, então o arquivo será criado.
- Se você deseja adicionar ao final do arquivo, deve usar o modo “a”.
- Arquivos já existentes só podem ser abertos para operações de leitura.
- Se o arquivo não existe, um erro é devolvido.
- Se um arquivo é aberto para operações de leitura/escrita, ele não será apagado caso já exista e, senão existir, ele será criado.



# Fechando um Arquivo

- A função `fclose()` fecha uma stream que foi aberta por meio de uma chamada a `fopen()`
- Ela escreve qualquer dado que ainda permanece no *buffer* de disco no arquivo e, então, fecha normalmente o arquivo em nível de sistema operacional
- Se um arquivo é aberto para operações de leitura/escrita, ele não será apagado caso já exista e, senão existir, ele será criado.
- Um `fclose()` também libera o bloco de controle de arquivo associado à *stream*, deixando-o disponível para reutilização.
- Em muitos casos, há um limite do sistema operacional para o número de arquivos abertos simultaneamente, assim, você deve fechar um arquivo antes de abrir outro.





- A função `fclose()` tem o seguinte protótipo:

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

onde `fp` é o ponteiro de arquivo devolvido pela chamada a `fopen()`.

- Um valor de retorno zero significa uma operação de fechamento bem sucedida.
- Qualquer outro valor indica um erro.
- A função padrão `ferror()` pode ser utilizada para determinar e informar qualquer problema.
- Geralmente, `fclose()` falhará quando um disco tiver sido retirado prematuramente do acionador ou não houver mais espaço no disco.



## Veja

- exemplo-53.c
- exemplo-54.c
- exemplo-55.c
- exemplo-56.c
- Assista os vídeos da playlist disponível em [http://bit.ly/pc\\_arq](http://bit.ly/pc_arq)

- 1 Modifique o programa `exemplo-48.c`, de modo a incluir duas novas opções no menu principal. Gravar Lista de Endereços em Arquivo e Recuperar Lista de Endereços de Arquivo. Implemente estas novas rotinas, sempre solicitando ao usuário que informe o nome do arquivo para cada operação. Faça testes manipulando tanto com arquivos binários quanto com arquivos texto.

