

# PROGRAMAÇÃO COMPUTACIONAL PARA ENGENHARIA

## MANIPULAÇÃO DE ARQUIVOS

Maurício Moreira Neto<sup>1</sup>

<sup>1</sup> **Universidade Federal do Ceará**  
**Departamento de Computação**

30 de janeiro de 2020

# Sumário

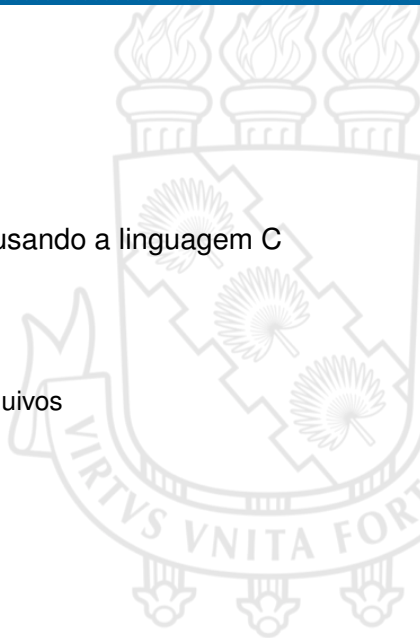
- 1 Manipulação
- 2 Abrir Arquivo
- 3 Fechando Arquivo
- 4 Escrita/Leitura

- 5 Fim-Arquivo
- 6 String
- 7 Blocos
- 8 Fluxo
- 9 Movendo



# Objetivos

- Aprender a manipular arquivos usando a linguagem C
  - Leitura
  - Escrita
  - Manipulação de dados em arquivos



# Arquivos

- Por que usar arquivos?
  - Permitem armazenar grande quantidade de informação
  - Persistência dos dados (em disco)
  - Acesso aos dados poder ser não sequencial
  - Acesso concorrente aos dados (mais de um programa pode usar os dados ao mesmo tempo)

# Tipos de Arquivos

- Basicamente, a linguagem C trabalha com dois tipos de arquivos: de texto e binários
- Arquivo texto
  - Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples como o Bloco de Notas
  - Os dados são gravados como caracteres de 8 bits
    - **Exemplo:** Um número inteiro de 32 bits com 8 dígitos ocupará 64 bits no arquivo (8 bits por dígito)

# Tipos de Arquivos

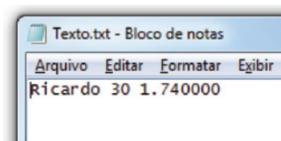
## ■ Arquivo binário

- Armazena uma sequência de bits que está sujeita as convenções dos programas que o gerou
  - **Exemplo:** arquivos executáveis, arquivos compactados, arquivos de registros, etc
- Os dados são gravados na forma binária (do mesmo modo que estão na memória)
  - **Exemplo:** um número inteiro de 32 bits com 8 dígitos ocupará 32 bits no arquivo

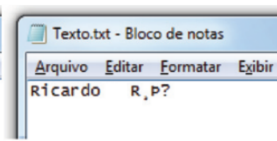
# Tipos de Arquivos

- **Exemplo:** Os dois trechos de arquivo abaixo possuem os mesmo dados

```
char nome[20] = "Ricardo";  
int i = 30;  
float a = 1.74;
```



Arquivo Texto



Arquivo Binário

# Manipulando Arquivos

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: **stdio.h**

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



# Manipulando Arquivos

- A linguagem C não possui funções que, automaticamente, leiam todas as informações de um arquivo
  - Suas funções se limitam a abrir/fechar e ler caracteres/bytes
  - É tarefa do programador criar a função que irá ler um arquivo de uma maneira específica

# Manipulando Arquivos

- Todas as funções de manipulação de arquivos trabalham com o conceito de “ponteiro de arquivo”
- Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

- **p** é o ponteiro para arquivos que nos permitirá manipular arquivos no C

# Abrindo um Arquivo

- Para a abertura de um arquivo, usa-se a função **fopen**

```
FILE *fopen(char *nome_arquivo, char *modo);
```

- O parâmetro **nome\_arquivo** determina qual arquivo deverá ser aberto, sendo que o mesmo deve ser válido no sistema operacional que estiver sendo utilizado

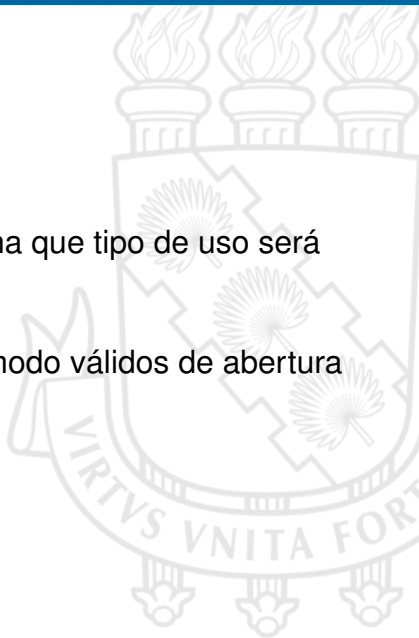
# Abrindo um Arquivo

- No parâmetro **nome\_arquivo** pode-se trabalhar com caminhos absolutos ou relativos
  - **Caminho absoluto:** descrição de um caminho desde o diretório raiz
    - **Exemplo** – C:\ \Projetos\ \dados.txt
  - **Caminho relativo:** descrição de um caminho desde o diretório corrente (onde o programa está salvo)
    - arq.txt
    - ..\ \dados.txt

```
FILE *fopen(char *nome_arquivo, char *modo);
```

# Abrindo um Arquivo

- 1 O modo de abertura determina que tipo de uso será feito do arquivo
- 2 A tabela a seguir mostra os modo válidos de abertura de um arquivo



# Abrindo um Arquivo

| Modo  | Arquivo | Função  |
|-------|---------|---|
| "r"   | Texto   | Leitura. Arquivo deve existir.  |
| "w"   | Texto   | Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.         |
| "a"   | Texto   | Escrita. Os dados serão adicionados no fim do arquivo ("append").             |
| "rb"  | Binário | Leitura. Arquivo deve existir.  |
| "wb"  | Binário | Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.         |
| "ab"  | Binário | Escrita. Os dados serão adicionados no fim do arquivo ("append").             |
| "r+"  | Texto   | Leitura/Escrita. O arquivo deve existir e pode ser modificado.                |
| "w+"  | Texto   | Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir. |
| "a+"  | Texto   | Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").     |
| "r+b" | Binário | Leitura/Escrita. O arquivo deve existir e pode ser modificado.                |
| "w+b" | Binário | Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir. |
| "a+b" | Binário | Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").     |

# Abrindo um Arquivo

- Um arquivo binário pode ser aberto para escrita utilizando o seguinte conjunto de comandos:
  - A condição **fp==NULL** testa se o arquivo foi aberto com sucesso. No caso de erro a função **fopen()** retorna um ponteiro nulo (**NULL**)

```
int main() {  
    FILE *fp;  
    fp = fopen("exemplo.bin", "wb");  
    if (fp == NULL)  
        printf("Erro na abertura do arquivo.\n");  
  
    fclose(fp);  
  
    return 0;  
}
```

# Erro ao Abrir um Arquivo

- Caso o arquivo não tenha sido aberto com sucesso
  - Provavelmente o programa não poderá continuar a executar
  - Nesse caso, utilizamos a função **exit()**, presente na biblioteca **stdlib.h**, para abortar o programa

```
void exit(int codigo_retorno);
```



# Erro ao Abrir um Arquivo

- A função **exit()** 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
  - um número **diferente de zero**, no caso de ter ocorrido um problema

# Erro ao Abrir um Arquivo

## ■ Exemplo:

```
int main() {  
    FILE *fp;  
    fp = fopen("exemplo.bin", "wb");  
    if(fp == NULL) {  
        printf("Erro na abertura do arquivo\n");  
        system("pause");  
        exit(1);  
    }  
    fclose(fp);  
  
    return 0;  
}
```

# Posição do Arquivo

- Ao se trabalhar com arquivos, existe uma espécie de posição onde estamos dentro do arquivo. É nessa posição onde será lido ou escrito o próximo caractere
  - Quando utilizando o acesso sequencial, raramente é necessário modificar essa posição
  - Isso por que, quando lemos um caractere, a posição no arquivo é automaticamente atualizada
  - Leitura e escrita em arquivos são parecidos com escrever em uma **máquina de escrever**

# Fechando o Arquivo

- Sempre que terminamos de usar um arquivo que abrimos, devemos fechá-lo. Para isso usa-se a função **fclose()**
- O ponteiro **fp** passado à função **fclose()** determina o arquivo a ser fechado. A função retorna zero no caso de sucesso

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

# Fechando o Arquivo

- Por que devemos fechar o arquivo?
  - Ao fechar um arquivo, todo caractere que tenha permanecido no “buffer” é gravado
  - O “buffer” é uma região de memória que armazena temporariamente os caracteres a serem gravados em disco imediatamente. Apenas quando o “buffer” está cheio é que seu conteúdo é escrito no disco

# Fechando o Arquivo

**Por que utilizar um “buffer”??**



# Fechando o Arquivo

Por que utilizar um “buffer”??

**Eficiência!!**



# Fechando o Arquivo

- Para ler e escrever arquivos no disco temos que posicionar a cabeça de gravação em um ponto específico do disco
- Se tivéssemos que fazer isso para cada caractere lido/escrito, a leitura/escrita de um arquivo seria uma operação muito lenta
- Assim a gravação só é realizada quando há um volume razoável de informações a serem gravadas ou quando o arquivo for fechado
- A função **exit()** fecha todos os arquivos que um programa tiver aberto



# Escrita/Leitura em Arquivo

- Uma vez aberto um arquivo, podemos ler ou escrever nele
- Para tanto, a linguagem C conta com uma série de funções de leitura/escrita que variam de funcionalidade para atender as diversas aplicações

# Escrita/Leitura de Caracteres

- A maneira mais fácil de se trabalhar com um arquivo é a leitura/escrita de um único caractere
- A função mais básica de entrada de dados é a função **fputc** (put character)

```
int fputc (int ch, FILE *fp);
```

- Cada invocação dessa função grava um único caractere **ch** no arquivo especificado por **fp**

# Escrita/Leitura de Caracteres

## ■ Exemplo da função `fputc`

```
int main(){
    FILE *arq;
    char string[100];
    int i;
    arq = fopen("arquivo.txt", "w");
    if(arq == NULL){
        printf("Erro na abertura do arquivo");
        system("pause");
        exit(1);
    }
    printf("Entre com a string a ser gravada no arquivo:");
    gets(string);
    //Grava a string, caractere a caractere
    for(i = 0; i < strlen(string); i++){
        fputc(string[i], arq);
    }
    fclose(arq);

    return 0;
}
```



# Escrita/Leitura de Caracteres

- A função **fputc** também pode ser utilizada para escrever um caractere na tela
  - Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:
  - Por exemplo: **fputc ('\*', stdout)** exibe um \* na tela do monitor (dispositivo de saída padrão)

```
int main() {  
    fputc ('*', stdout);  
  
    return 0;  
}
```

# Escrita/Leitura de Caracteres

- Da mesma maneira que gravamos um único caractere no arquivo, também podemos ler um único caractere
- A função correspondente de leitura de caracteres é **fgetc** (get character)

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

# Escrita/Leitura de Caracteres

- Cada chamada da função **fgetc** lê um único caractere do arquivo especificado
  - Se **fp** aponta para um arquivo, então **fgetc(fp)** lê o caractere atual no arquivo e se posiciona para ler o próximo caractere do arquivo
  - **Lembre-se, a leitura em arquivos é parecida com escrever em uma máquina de escrever**

```
char c;  
c = fgetc (fp);
```

# Escrita/Leitura de Caracteres

## ■ Exemplo da função `fgetc`

```
int main() {  
    FILE *arq;  
    char c;  
    arq = fopen("arquivo.txt", "r");  
    if(arq == NULL) {  
        printf("Erro na abertura do arquivo");  
        system("pause");  
        exit(1);  
    }  
    int i;  
    for(i = 0; i < 5; i++) {  
        c = fgetc(arq);  
        printf("%c", c);  
    }  
    fclose(arq);  
    return 0;  
}
```



# Escrita/Leitura de Caracteres

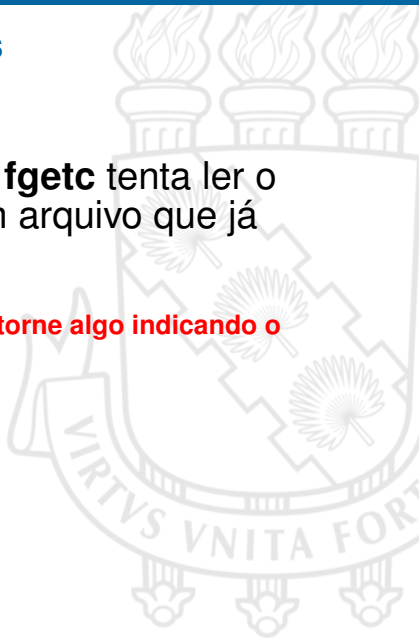
- Similar ao que acontece com a função **fputc**, a função **fgetc** também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão):
- Nesse caso, **fgetc(stdin)** lê o próximo caractere digitado no teclado

```
int main() {  
    char ch;  
    ch = fgetc(stdin);  
  
    printf("%c\n", ch);  
  
    return 0;  
}
```



# Escrita/Leitura de Caracteres

- O que acontece quando **fgetc** tenta ler o próximo caractere de um arquivo que já acabou?
  - **Precisamos que a função retorne algo indicando o arquivo acabou, correto?**



# Escrita/Leitura de Caracteres

- O que acontece quando **fgetc** tenta ler o próximo caractere de um arquivo que já acabou?
  - **Precisamos que a função retorne algo indicando o arquivo acabou, correto?**
- Porém, todos os 256 caracteres são “válidos”!

# Escrita/Leitura de Caracteres

- Para evitar esse tipo de situação, **fgetc** não devolve um **char** mas um **int**:

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

- O conjunto de valores do **char** está contido dentro do conjunto do **int**
  - Se o arquivo tiver acabado, **fgetc** devolve um valor **int** que não possa ser confundido com um **char**

# Escrita/Leitura de Caracteres

- Assim, se o arquivo não tiver mais caracteres, **fgetc** devolve o valor  $-1$
- A **fgetc** devolve a constante **EOF** (*end of file*), que está definida na biblioteca **stdio.h**. Em muitos computadores o valor de **EOF** é  $-1$

```
char c;  
c = fgetc(fp);  
if (c == EOF)  
    printf ("O arquivo terminou!\n");
```

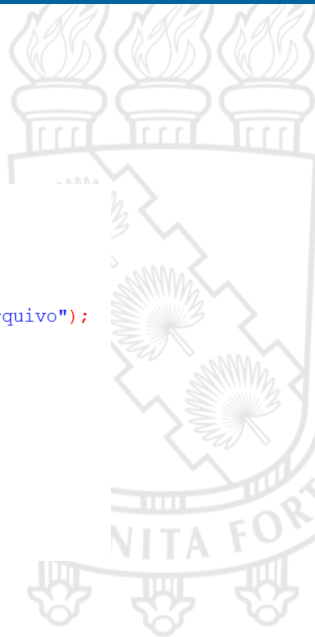
# Escrita/Leitura de Caracteres

## ■ Exemplo de uso do EOF

```
int main() {
    FILE *arq;
    char c;
    arq = fopen("arquivo.txt", "r");
    if(arq == NULL) {
        printf("Erro na abertura do arquivo");
        system("pause");
        exit(1);
    }
    while((c = fgetc(arq)) != EOF)
        printf("%c", c);

    fclose(arq);

    return 0;
}
```



# Fim do Arquivo

- Como visto, **EOF** (“End of file”) indica o fim de um arquivo
- No entanto, podemos também utilizar a função **feof** para verificar se um arquivo chegou ao fim, cujo protótipo é

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

- No entanto, é muito comum fazer mau uso dessa função!

# Fim do Arquivo

- Um mau uso muito comum da função **feof()** é usá-la para terminar um loop
  - Mas por que isso é um mau uso?

```
int main{
    int i, n;
    FILE *F = fopen("teste.txt", "r");
    if(arq == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    while(!feof(F)){
        fscanf(F, "%d", &n);
        printf("%d\n", n);
    }
    fclose(F);

    return 0;
}
```

# Fim do Arquivo

- Vamos ver a descrição da função **feof()**
  - A função **feof()** testa o indicador de fim de arquivo para o fluxo apontado por **fp**
  - A função retorna um valor inteiro **diferente de zero** se, e somente se, **o indicador de fim de arquivo** está marcado para **fp**
- Ou seja, a função testa o **indicador de fim de arquivo**, não o próprio **arquivo**

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



# Fim do Arquivo

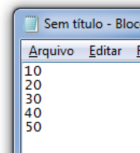
- Isso significa que outra função é responsável por alterar o indicador para indicar que o **EOF** foi alcançado
  - A maioria das funções de leitura irá alterar o indicador após ler todos os dados, e então realizar uma nova leitura resultando em nenhum dado, apenas o **EOF**
- Como resolver isso ?

# Fim do Arquivo

- Isso significa que outra função é responsável por alterar o indicador para indicar que o **EOF** foi alcançado
  - A maioria das funções de leitura irá alterar o indicador após ler todos os dados, e então realizar uma nova leitura resultando em nenhum dado, apenas o **EOF**
- Como resolver isso ?
  - Devemos evitar o uso da função **feof()** para testar um loop e usá-la para testar se uma leitura alterou o **indicador de fim de arquivo**

# Fim do Arquivo

- Para entender esse problema do mau uso da funções **feof()**, considere que queiramos ler todos os números contidos em um arquivo texto como o mostrado abaixo



# Fim do Arquivo

## Mau uso da função feof()

```
int main{
    int i, n;
    FILE *F = fopen("teste.txt", "r");
    if(arq == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    while(!feof(F)){
        fscanf(F, "%d", &n);
        printf("%d\n", n);
    }
    fclose(F);

    return 0;
}
```

Saída: 10 20 30 40 50 50

## Bom uso da função feof()

```
int main{
    int i, n;
    FILE *F = fopen("teste.txt", "r");
    if(arq == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    while(1){
        fscanf(F, "%d", &n);
        if(feof(F))
            break;
        printf("%d ", n);
    }
    fclose(F);

    return 0;
}
```

Saída: 10 20 30 40 50

# Arquivos pré-definidos

- Como visto anteriormente, os ponteiros **stdin** e **stdout** podem ser utilizados para acessar os dispositivos de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão
- No início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles **stdin** e **stdout**

# Arquivos pré-definidos

## ■ Alguns arquivos pré-definidos

### ■ **stdin**

- Dispositivo de entrada padrão (geralmente o teclado)

### ■ **stdout**

- Dispositivo de saída padrão (geralmente a tela do monitor)

### ■ **stderr**

- Dispositivo de saída de erro padrão (geralmente a tela do monitor)

### ■ **stdaux**

- Dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial)

### ■ **stdprn**

- Dispositivo de impressão padrão (em muitos sistemas, associado à porta paralela)

# Escrita/Leitura de Strings

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo
- Porém, existem funções na linguagem C que permitem ler/escrever uma sequência de caracteres, isto é, uma string!
  - `fputs()`
  - `fgets()`

# Escrita/Leitura de Strings

- Basicamente, para se escrever uma string em um arquivo usamos a função **fputs**:

```
int fputs(char *str, FILE *fp);
```

- Esta função recebe como parâmetro um array de caracteres (string) e um ponteiro para o arquivo no qual queremos escrever



# Escrita/Leitura de Strings

- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro **diferente de zero** é retornado
  - Se houver erro na escrita, o valor **EOF** é retornado
- Como a função **fputc**, **fputs** também pode ser utilizada para escrever uma string na tela:

```
int main(){  
    char texto[30] = "Hello World\n";  
    fputs(texto, stdout);  
  
    return 0;  
}
```

# Escrita/Leitura de Strings

## ■ Exemplo da função `fputs`:

```
int main() {
    char str[20] = "Hello World!";
    int result;
    FILE *arq;
    arq = fopen("ArqGrav.txt", "w");
    if(arq == NULL) {
        printf("Problemas na CRIACAO do arquivo\n");
        system("pause");
        exit(1);
    }
    result = fputs(str, arq);
    if(result == EOF)
        printf("Erro na Gravacao\n");
    fclose(arq);

    return 0;
}
```

# Escrita/Leitura de Strings

- Da mesma maneira que gravamos uma cadeia de caracteres no arquivo, a sua leitura também é possível
- Para se ler uma string de um arquivo podemos usar a função **fgets()** cujo protótipo é:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

# Escrita/Leitura de Strings

- A função **fgets** recebe 3 parâmetros
  - **str**: aonde a lida será armazenada, **str**
  - **tamanho**: o número máximo de caracteres a serem lidos
  - **fp**: ponteiro que está associado ao arquivo de onde a string será lida
  
- Retorno:
  - NULL em caso de erro ou fim do arquivo
  - O ponteiro para o primeiro caractere recuperado em **str**

```
char *fgets(char *str, int tamanho, FILE *fp);
```

# Escrita/Leitura de Strings

## ■ Funcionamento da função **fgets**

- A função lê a string até que um caractere de nova linha seja lido ou *tamanho-1* caracteres tenham sido lidos
- Se o caractere de nova linha ('\n') for lido, ele fará parte da string, o que não acontecia com **gets**
- A string resultante sempre terminará com '\0' (por isto somente *tamanho-1* caracteres, no máximo, serão lidos)
- Se ocorrer algum erro, a função devolverá um ponteiro nulo em **str**

# Escrita/Leitura de Strings

- A função **fgets** é semelhante à função **gets**, porém, com as seguintes vantagens:
  - Pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha “\n” na string
  - Específica o tamanho máximo da string de entrada. Isso evita estouro no buffer

# Escrita/Leitura de Strings

## ■ Exemplo da função `fgets`

```
int main() {  
    char str[20];  
    char *result;  
    FILE *arq;  
    arq = fopen("ArqGrav.txt", "r");  
    if(arq == NULL) {  
        printf("Problemas na ABERTURA do arquivo\n");  
        system("pause");  
        exit(1);  
    }  
    result = fgets(str, 13, arq);  
    if(result == NULL)  
        printf("Erro na leitura\n");  
    else  
        printf("%s", str);  
    fclose(arq);  
  
    return 0;  
}
```

# Escrita/Leitura de Strings

- Vale lembrar que o ponteiro **fp** pode ser substituído por **stdin**, para se fazer a leitura do teclado:

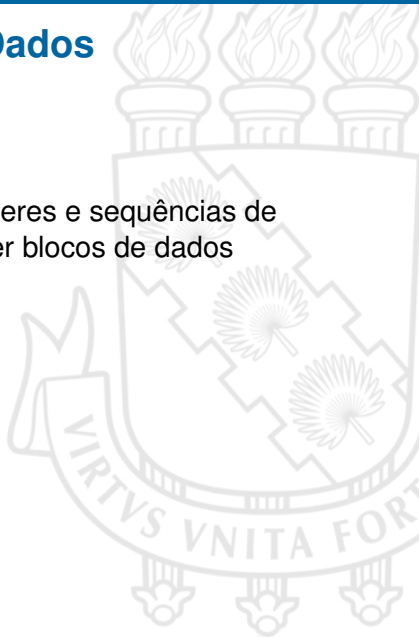
```
int main(){
    char nome[30];
    printf("Digite um nome: ");
    fgets(nome, 30, stdin);
    printf("O nome digitado foi: %s", nome);

    return 0;
}
```



# Escrita/Leitura de Bloco de Dados

- Além da leitura/escrita de caracteres e sequências de caracteres, podemos ler/escrever blocos de dados
- Para isso, temos duas funções:
  - `fwrite()`
  - `fread()`



# Escrita/Leitura de Bloco de Dados

- A função **fwrite** é responsável pela escrita de um bloco de dados da memória em um arquivo
- Seu protótipo é:

```
unsigned fwrite(void *buffer, int tamanho_bytes, int  
               count, FILE *fp);
```

# Escrita/Leitura de Bloco de Dados

- A função **fwrite** recebe 4 argumentos
  - **buffer**: ponteiro para a região de memória na qual estão os dados
  - **numero\_bytes**: tamanho de cada posição de memória a ser escrita
  - **count**: total de unidades de memória que devem ser escritas
  - **fp**: ponteiro associado ao arquivo onde os dados serão escritos

```
unsigned fwrite(void *buffer, int tamanho_bytes, int  
count, FILE *fp);
```

# Escrita/Leitura de Bloco de Dados

- Note que temos dois valores numéricos
  - `numero_bytes`
  - `count`
- Isto significa que o número total de bytes escritos é:
  - `numero_bytes * count`
- Como retorno, temos o número de unidades efetivamente escritas
  - Este número pode ser menor que **count** quando ocorrer algum erro.

# Escrita/Leitura de Bloco de Dados

## ■ Exemplo da função `fwrite`

```
int main() {
    FILE *arq;
    arq = fopen("ArqGrav.txt", "wb");

    char str[20] = "Hello World!";
    float x = 5;
    int v[5] = {1,2,3,4,5};
    //grava a string toda no arquivo
    fwrite(str, sizeof(char), strlen(str), arq);
    //grava apenas os 5 primeiros caracteres da string
    fwrite(str, sizeof(char), 5, arq);
    //grava o valor de x no arquivo
    fwrite(&x, sizeof(float), 1, arq);
    //grava todo o array no arquivo (5 posições)
    fwrite(v, sizeof(int), 5, arq);
    //grava apenas as 2 primeiras posições do array
    fwrite(v, sizeof(int), 2, arq);
    fclose(arq);

    return 0;
}
```

# Escrita/Leitura de Bloco de Dados

- A função **fread** é responsável pela leitura de um bloco de dados de um arquivo
- Seu protótipo é:

```
unsigned fread(void *buffer, int tamanho_bytes, int  
count, FILE *fp);
```

# Escrita/Leitura de Bloco de Dados

- A função **fread** funciona como a sua companheira **fwrite**, porém lendo dados do arquivo
- Como na função **fwrite**, **fread** retorna o número de itens escritos. Este valor será igual a **count** a menos que ocorra algum erro

```
unsigned fread(void *buffer, int tamanho_bytes, int  
              count, FILE *fp);
```

# Escrita/Leitura de Bloco de Dados

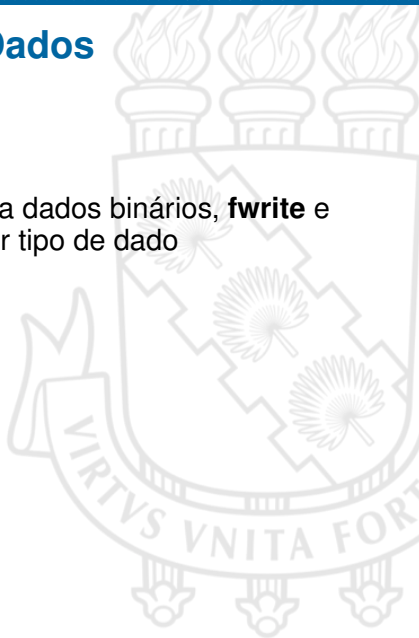
## ■ Exemplo da função `fread`

```
char str1[20],str2[20];
float x;
int i,v1[5],v2[2];
//lê a string toda do arquivo
fread(str1,sizeof(char),12,arq);
str1[12] = '\0';
printf("%s\n",str1);
//lê apenas os 5 primeiros caracteres da string
fread(str2,sizeof(char),5,arq);
str2[5] = '\0';
printf("%s\n",str2);
//lê o valor de x do arquivo
fread(&x,sizeof(float),1,arq);
printf("%f\n",x);
//lê todo o array do arquivo (5 posições)
fread(v1,sizeof(int),5,arq);
for(i = 0; i < 5; i++)
    printf("v1[%d] = %d\n",i,v1[i]);
//lê apenas as 2 primeiras posições do array
fread(v2,sizeof(int),2,arq);
for(i = 0; i < 2; i++)
    printf("v2[%d] = %d\n",i,v2[i]);
```



# Escrita/Leitura de Bloco de Dados

- Quando o arquivo for aberto para dados binários, **fwrite** e **fread** podem manipular qualquer tipo de dado
  - int
  - float
  - double
  - array
  - struct
  - ...



# Escrita/Leitura de Bloco de Dados

- As funções de fluxos padrão permitem ao programador ler e escrever em arquivos da maneira padrão com a qual o já líamos e escrevíamos na tela
- As funções **fprintf** e **fscanf** funcionam de maneiras semelhantes a **printf** e **scanf**, respectivamente
- A diferença é que elas direcionam os dados para arquivos

# Escrita/Leitura por Fluxo Padrão

## ■ Exemplo: **fprintf**

```
printf("Total = %d: ", x); //escreve na tela  
fprintf(fp, "Total = %d", x); //grava no arquivo fp
```

## ■ Exemplo: **fscanf**

```
fscanf("%d", &x); //lê do teclado  
fscanf(fp, "%d", &x); //lê do arquivo fp
```

# Escrita/Leitura por Fluxo Padrão

## ■ Atenção

- Embora **fprintf** e **fscanf** sejam mais fáceis de ler/escrever dados em arquivos, nem sempre elas são as escolhas mais apropriada
- Como os dados são escritos em ASCII e formatados como apareceriam em tela, um tempo extra é perdido
- Se a intenção é velocidade ou tamanho do arquivo, deve-se utilizar as funções **fread** e **fwrite**

# Escrita/Leitura por Fluxo Padrão

## ■ Exemplo da funções `fprintf`

```
int main() {
    FILE *arq;
    char nome[20] = "Ricardo";
    int I = 30;
    float a = 1.74;
    int result;
    arq = fopen("ArqGrav.txt", "w");
    if(arq == NULL) {
        printf("Problemas na ABERTURA do arquivo\n");
        system("pause");
        exit(1);
    }
    fprintf(arq, "Nome: %s\n", nome);
    fprintf(arq, "Idade: %d\n", i);
    fprintf(arq, "Altura: %f\n", a);
    fclose(arq);

    return 0;
}
```

# Escrita/Leitura por Fluxo Padrão

## ■ Exemplo da funções `fscanf`

```
int main(){
    FILE *arq;
    char texto[20], nome[20];
    int i;
    float a;
    int result;
    arq = fopen("ArqGrav.txt", "r");
    if(arq == NULL) {
        printf("Problemas na ABERTURA do arquivo\n");
        system("pause");
        exit(1);
    }
    fscanf(arq, "%s%s", texto, nome);
    printf("%s %s\n", texto, nome);
    fscanf(arq, "%s %d", texto, &i);
    printf("%s %d\n", texto, i);
    fscanf(arq, "%s%f", texto, &a);
    printf("%s %f\n", texto, a);
    fclose(arq);

    return 0;
}
```

# Movendo-se pelo Arquivo

- De modo geral, o acesso a um arquivo é sequencial. Porém, é possível fazer buscas e acessos randômicos em arquivos
- Para isso, existe a função **fseek**:

```
int fseek(FILE *fp, long numbytes, int origem);
```

- Basicamente, esta função move a posição corrente de leitura ou escrita no arquivo em tantos bytes, a partir de um ponto especificado

# Movendo-se pelo Arquivo

- A função **fseek** recebe 3 parâmetros
  - **fp**: o ponteiro para o arquivo
  - **numbytes**: é o total de bytes a partir de **origem** a ser pulado
  - **origem**: determina a partir de onde os **numbytes** de movimentação serão contados
  
- A função devolve o valor 0 quando bem sucedida

```
int fseek(FILE *fp, long numbytes, int origem);
```



# Movendo-se pelo Arquivo

- Os valores possíveis para **origem** são definidos por macros em **stdio.h** e são:

| Nome     | Valor | Significado               |
|----------|-------|---------------------------|
| SEEK_SET | 0     | Início do arquivo         |
| SEEK_CUR | 1     | Ponto corrente do arquivo |
| SEEK_END | 2     | Fim do arquivo            |

- Portanto, para mover **numbytes** a partir
  - Início do arquivo, **origem** deve ser **SEEK\_SET**
  - Posição atual, **origem** deve ser **SEEK\_CUR**
  - Final do arquivo, **origem** deve ser **SEEK\_END**
- numbytes** pode ser negativo quando usado com **SEEK\_CUR** e **SEEK\_END**

# Movendo-se pelo Arquivo

## ■ Exemplo da função `fseek`

```
struct cadastro{ char nome[20], rua[20]; int idade;};
int main(){
    FILE *f = fopen("arquivo.txt", "wb");

    struct cadastro c, cad[4] = {"Ricardo", "Rua 1", 31,
                                "Carlos", "Rua 2", 28,
                                "Ana", "Rua 3", 45,
                                "Bianca", "Rua 4", 32};

    fwrite(cad, sizeof(struct cadastro), 4, f);
    fclose(f);

    f = fopen("arquivo.txt", "rb");
    fseek(f, 2*sizeof(struct cadastro), SEEK_SET);
    fread(&c, sizeof(struct cadastro), 1, f);
    printf("%s\n%s\n%d\n", c.nome, c.rua, c.idade);
    fclose(f);

    return 0;
}
```

# Movendo-se pelo Arquivo

- Outra opção de movimentação pelo arquivo é simplesmente retornar para o seu início
- Para tanto, usa-se a função **rewind**:

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

# Apagando um Arquivo

- Além de permitir manipular arquivos, a linguagem C também permite apagá-lo do disco. Isso pode ser feito utilizando a função **remove**:

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

- Diferente das funções vistas até aqui, esta função recebe o **caminho** e **nome** do arquivo a ser excluído, e não um ponteiro para FILE
- Como retorno temos um valor inteiro, o qual será igual a 0 se o arquivo for excluído com sucesso

# Apagando um Arquivo

## ■ Exemplo da função `remove`

```
int main(){
    int status;
    status = remove("ArgGrav.txt");
    if(status != 0){
        printf("Erro na remocao do arquivo.\n");
        system("pause");
        exit(1);
    }else
        printf("Arquivo removido com sucesso.\n");

    return 0;
}
```

# Referências

- André Luiz Villar Forbellone, Henri Frederico Eberspächer, **Lógica de programação** (terceira edição), Pearson, 2005, ISBN 9788576050247.
- Ulysses de Oliveira, **Programando em C - Volume I - Fundamentos**, editora Ciência Moderna, 2008, ISBN 9788573936599
- **Slides baseados no material do site “Linguagem C Descomplicado”**
  - <https://programacaodescomplicada.wordpress.com/complementar/>