

# Algoritmos II



ARQUIVOS

# Conceito



- Até agora quase todos os programas solicitavam ao usuário dados para serem manipulados pelo programa.
- Uma vez o programa terminado, todos os dados introduzidos, ou resultantes do programa eram perdidos, pois não eram armazenados de forma definitiva em um repositório permanente
- Em C um arquivo é apenas um conjunto de bytes colocados uns após os outros sequencialmente

# Conceito



- Arquivos podem ser vistos em dois sentidos distintos:
  - Quando o arquivo é a fonte de dados para o programa
    - ✦ Neste caso trata-se de um conjunto de entrada de dados (input)
  - Quando o arquivo é o destino dos dados gerados pelo programa
    - ✦ Neste caso trata-se de um arquivo de saída de dados (output)

# Tipos de periféricos



- Os dados de entrada podem ser fornecidos através de periféricos de entrada como o teclado, mouse, leitor de códigos de barras, etc.
- Os dados de saída podem ser enviados através de periféricos de saída, como a tela do computador, impressora, etc.
- Existem, no entanto periféricos que comportam ambas as características, isto é, permitem entrada e saída de dados.
  - Como exemplo temos o disco rígido do computador, que permite leitura e escrita de dados.
- De qualquer forma, independente do tipo de periférico que se esteja utilizando em C, todas as entradas e saídas de dados são processadas através de streams.

# Streams



- Um stream é um conjunto sequencial de caracteres, isto é, um conjunto de bytes sem qualquer estrutura interna.
- A maior vantagem na utilização de streams para entrada e saída de dados é que esta é realizada independente do periférico que esteja sendo utilizado (Device Independent)
- Evita-se com isso a escrita de porções de códigos específicas para envio dos mesmos dados a diferentes dispositivos.

# Operações básicas sobre arquivos



- **Abertura:** Para processar qualquer arquivo deve-se primeiramente “abrir o arquivo”. Esta operação é realizada através da associação de uma variável do programa ao arquivo que se pretende processar.
- **Leitura/Escrita:** Depois do arquivo aberto, podem ser realizadas a leitura, escrita, posicionamento ao longo do arquivo, etc.
- **Fechamento:** Ao final do processamento do arquivo deve ser retirada a ligação criada entre a variável e o arquivo (Fechamento do arquivo)

# Abertura de um arquivo



- Para utilização de um arquivo deve ser declarada uma variável do tipo FILE (ou, mais apropriadamente, um ponteiro para o tipo FILE)
- A declaração de uma variável do tipo FILE \* faz com que esta seja um ponteiro para o tipo FILE
- Sua declaração é feita tal como qualquer outra variável

```
int x, y;  
Float k;  
FILE *fp; /* fp - file pointer */
```

# Abertura de um arquivo



- A abertura do arquivo é realizada utilizando a função **fopen**, cujo protótipo se encontra na biblioteca `stdio.h`, pois se trata de uma operação padrão de entrada e saída:

```
FILE *fopen(const char *filename, const char *mode;
```

... **const** indica que os parâmetros não serão alterados dentro da função

- A função recebe assim 2 parâmetros:
  - **filename**: String contendo o nome físico do arquivo
    - ✦ Ex: “dados.dat”
  - **mode**: String contendo o modo de abertura do arquivo



# Abertura de um arquivo



- O nome de um arquivo é armazenado em uma string e deve representar fielmente o nome conforme é visto pelo SO.
- No entanto no MS-DOS os arquivos podem ser representados usando o caractere “\”, que é um caractere especial em C. Portanto, caso o nome do arquivo seja escrito no programa, deve-se colocar \\ para representar cada caractere.
- Exemplo:
  - `C:\tmp\dados.dat` deverá ser escrito como “`C:\\tmp\\dados.dat`”
- Caso a leitura seja realizada a partir do teclado deve-se colocar apenas uma \, pois a função de leitura sabe o que está processando

# Tipos de arquivo



- Existem basicamente dois tipos de arquivos em C, arquivos **Texto** e **Binários**.
- **Texto:** é constituído por caracteres que são perceptíveis por nós como letras do alfabeto, números e símbolos como &%\$#().;:, e ainda pelos separadores espaço em branco, tab, e NewLine.
- **Binário:** Podem gravar qualquer caractere da tabela ASCII, normalmente são utilizados para gravar informações mais complexas que textos legíveis. São exemplos de arquivos binários músicas (\*.mp3), vídeos (\*.avi) e imagens (\*.bmp).

# Abertura de um arquivo



Modo	Descrição
w	<ul style="list-style-type: none"><li>• Cria um <b>NOVO</b> arquivo vazio para <b>ESCRITA</b>.</li><li>• Caso um arquivo com o mesmo nome já existe ele será substituído por um arquivo vazio.</li></ul>
w+	<ul style="list-style-type: none"><li>• Cria um <b>NOVO</b> arquivo vazio para <b>LEITURA e ESCRITA</b>.</li><li>• Caso um arquivo com o mesmo nome já existe ele será substituído por um arquivo vazio.</li></ul>
r	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> somente para <b>LEITURA</b>.</li><li>• Caso não possa ser aberto (não existe ou já foi aberto por outro programa) a função retorna NULL.</li></ul>
r+	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> para <b>LEITURA e ESCRITA</b>.</li><li>• É necessário que o arquivo exista.</li><li>• Caso não possa ser aberto a função retorna NULL.</li></ul>
a	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> para <b>ESCRITA</b> no <b>FINAL do arquivo</b>.</li><li>• Não apaga o conteúdo pré-existente no arquivo. (O "a" vem do inglês <i>append</i>, adicionar, apender)</li></ul>
a+	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> para <b>ESCRITA</b> no <b>FINAL do arquivo e LEITURA</b>.</li></ul>

# Definição do Tipo de Arquivo a ser Aberto



- O modo de abertura ainda pode ser combinado com o tipo de arquivo considerando-o um **arquivo de texto (t)** ou um **arquivo binário (b)**.
- Por padrão, a abertura de um arquivo é realizada considerando este como um arquivo de texto. Para abrir um arquivo em modo binário é necessário acrescentar um **b** ao modo de abertura
  - Ex: “rb”, “wb”, “ab”, “a+b”, etc.
- Exemplos:
  - `fp = fopen("file.txt", "rb");`
    - ✦ → Leitura de arquivo binário
  - `fp = fopen("file.txt", "a+b");`
    - ✦ → Leitura e gravação no final do arquivo binário.

# Fechamento de um arquivo



- O fechamento de um arquivo remove a ligação entre a variável do programa e o arquivo existente no disco.
- Antes do arquivo ser fechado são gravados todos os dados que possam ainda existir em buffers associados aos arquivos.
- É liberada a memória alocada pela função `fopen` para a estrutura do tipo `FILE`.

```
Int fclose (FILE *arq)
```

# Gravação parcial da memória para o disco



- Existe ainda uma função que permite ao usuário gravar fisicamente dados que estejam em memória no buffer associado ao arquivo.
- A função chamada fflush:

```
Int fflush (FILE *arq)
```

- A função fflush grava todos os dados em arquivo, mas manter aberto o arquivo correspondente.

# Resumo de funções



fopen

- **Abre** ou **Cria** um arquivo texto ou binário
- Define operação de **Leitura** e/ou **Gravação**

???

- Serão apresentadas em seguida as operações de leitura e gravação.
- Serão apresentadas também funções de posicionamento no arquivo.

fflush

- Grava os dados em memória para o arquivo.
- Mantém o arquivo aberto para leitura e/ou gravação.
- Execução opcional.

fclose

- Grava os dados em memória para o arquivo.
- Fecha o arquivo.
- Nenhuma leitura ou gravação é permitida sem que o arquivo seja reaberto.

# Exemplo de abertura de um arquivo



- Abertura de arquivo texto para leitura

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char s[100];

    puts("Introduza o Nome do Arquivo: ");
    gets(s);
    fp = fopen(s, "r");

    if(fp==NULL)
        printf("Impossível abrir o arquivo %s\n",s);
    else
        printf("Arquivo %s aberto com sucesso!!!\n",s);
    fclose(fp);

    return 0;
}
```



# Exemplo de abertura de um arquivo



- Criação de novo arquivo texto

```
#include <stdio.h>

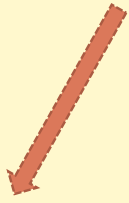
int main()
{
    FILE *fp;

    fp = fopen ("README.TXT", "w");

    if (fp == NULL) {
        printf ("Houve um erro ao criar o arquivo.\n");
        return 1;
    }
    printf ("Arquivo README criado com sucesso.\n");

    fclose (fp);

    return 0;
}
```



# Leitura de Caracteres de um Arquivo



- Dentre as diversas funções que permitem a leitura de arquivos, uma das mais utilizada é a **fgetc** (*file get char*).

```
Int fgetc (FILE *arq)
```

- A função lê **1 caractere** do arquivo “**arq**”, previamente aberto pela função **fopen**.
- Retorna o caractere lido ou **EOF** caso não existam caracteres a serem lidos (fim de arquivo).

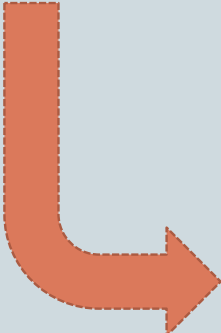
# Exemplo de Leitura de Arquivo



README.TXT

1	Arquivo exemplo
2	Exemplo de linha 1.
3	Exemplo de linha 2.
4	Exemplo de linha 3.

```
Arquivo exemplo
Exemplo de linha 1.
Exemplo de linha 2.
Exemplo de linha 3.
Process returned 0 (0x0)   execution time : 0.240 s
Press any key to continue.
```



```
#include <stdio.h>
int main()
{
    FILE *fp;
    char ch;

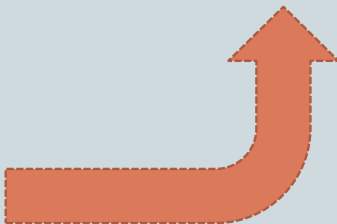
    fp = fopen ("README.TXT", "r");

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

    while (( ch=fgetc (fp) ) != EOF)
        putchar (ch) ;

    fclose (fp);

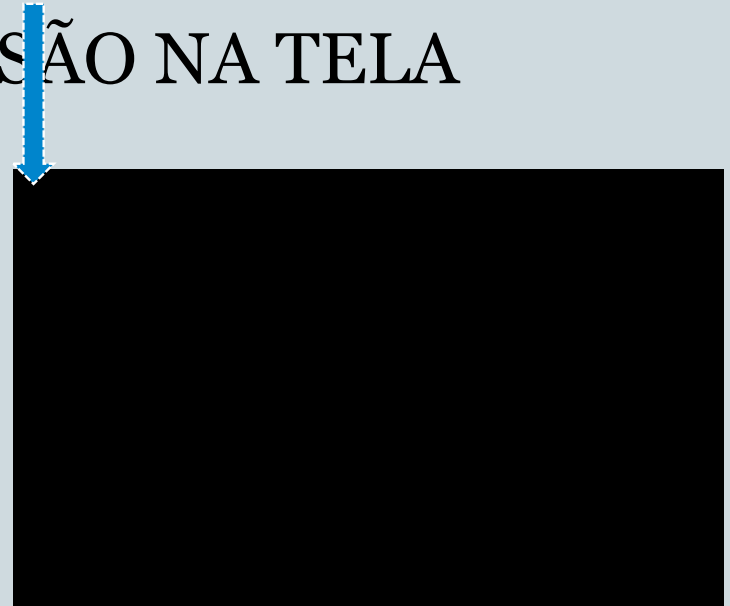
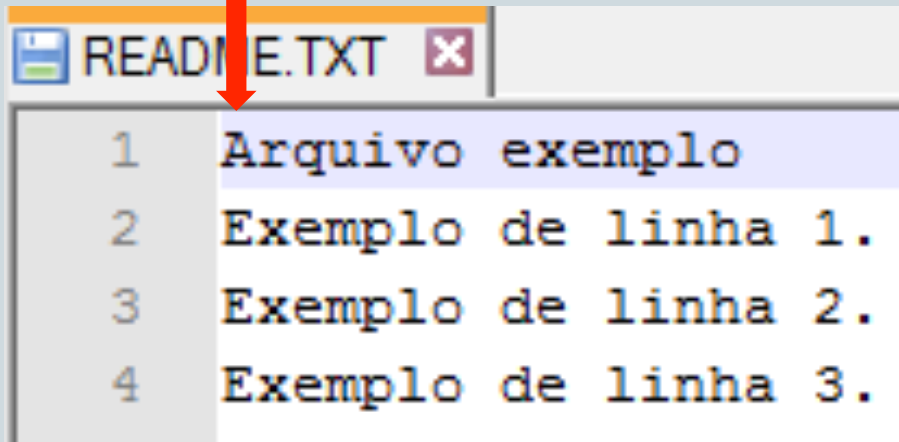
    return 0;
}
```



# fgetc Passo a Passo



- Ao abrir o arquivo, o comando **fopen** posiciona o cursor de leitura no início do arquivo.
- Seta Vermelha – Cursor de LEITURA
- Seta Azul – cursor de IMPRESSÃO NA TELA



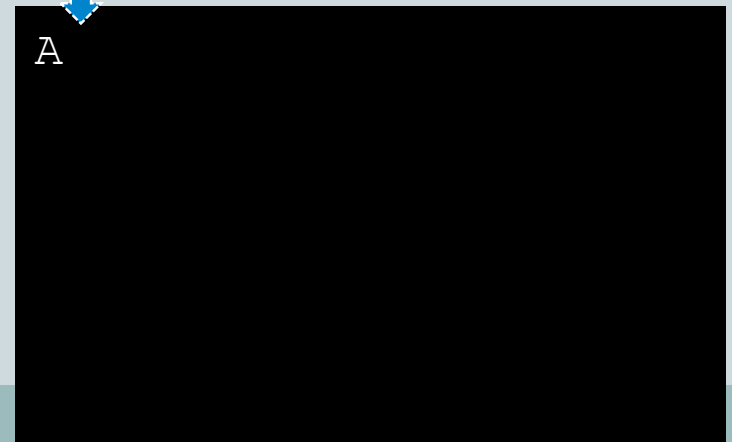
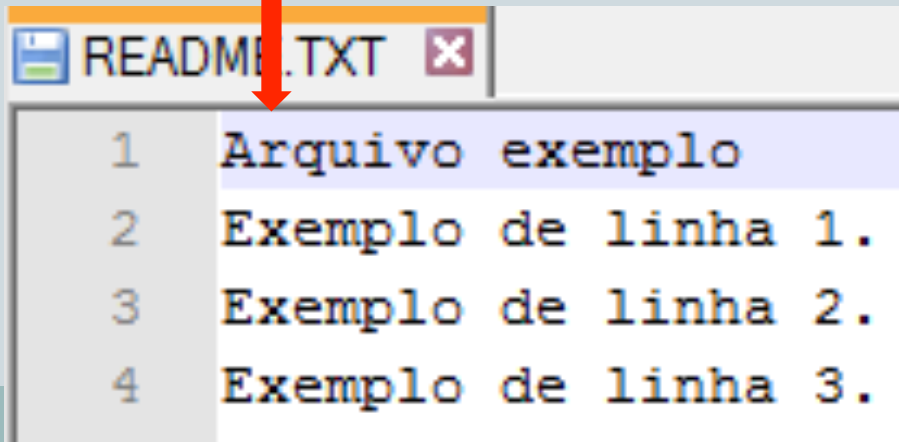
# fgetc Passo a Passo



- Ao executar o comando **fgetc** este retorna o caractere apontado pelo cursor de leitura, na imagem “A”.

```
while (( ch=fgetc(fp)) != EOF)
    putchar(ch);
```

- **O cursor de leitura é movimentado para o próximo caractere do arquivo.**
- Se o houve um caractere retornado por **fgetc**, ou seja, não é o final do arquivo (EOF), então o caractere é impresso.



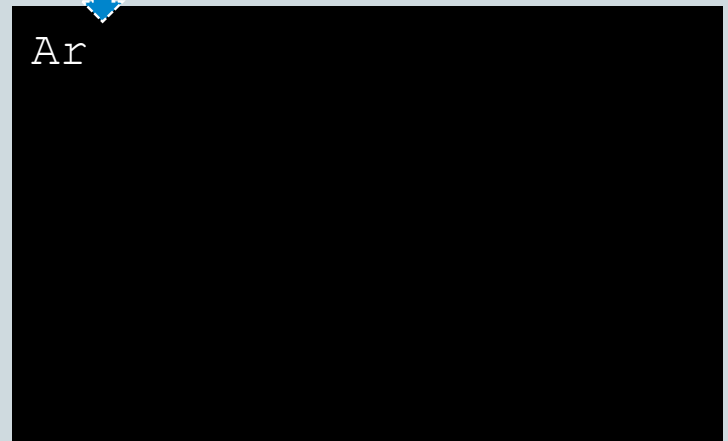
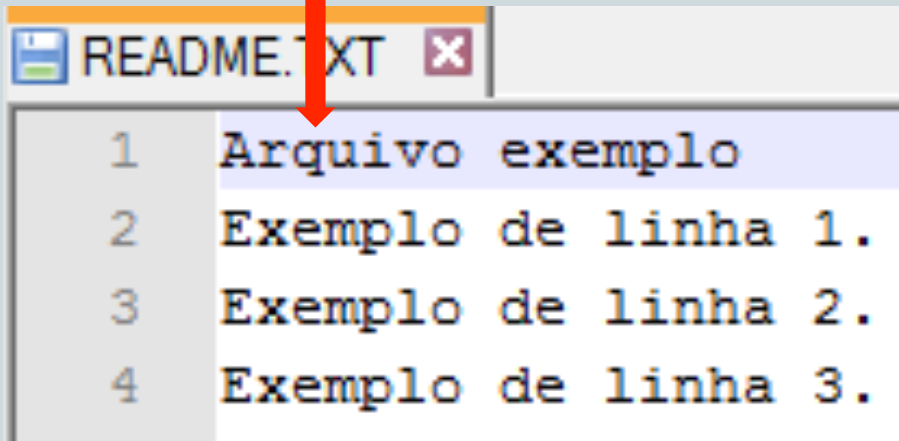
# fgetc Passo a Passo



- O laço **while** repete a leitura pelo comando **fgetc** enquanto não for final do arquivo (**EOF**).

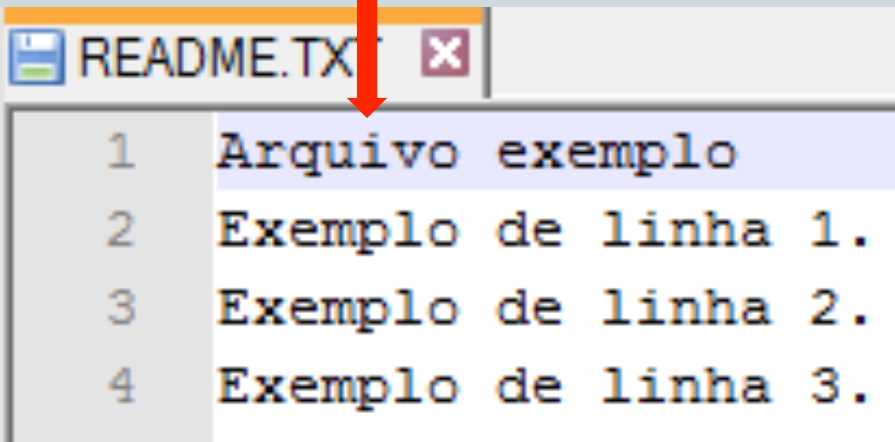
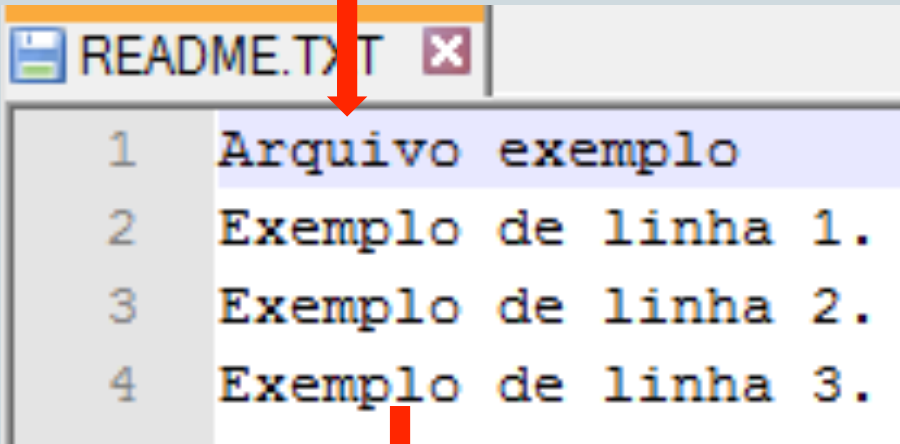
```
while (( ch=fgetc(fp)) != EOF)
    putchar(ch);
```

- O comando `putchar(ch)` imprime caractere a caractere lido.



# fgetc Passo a Passo

```
while (( ch=fgetc(fp)) != EOF)  
    putchar(ch);
```

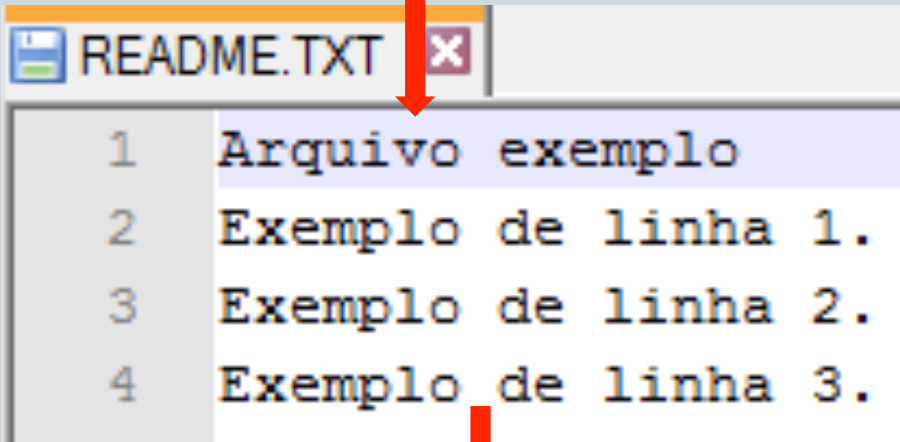


Arq

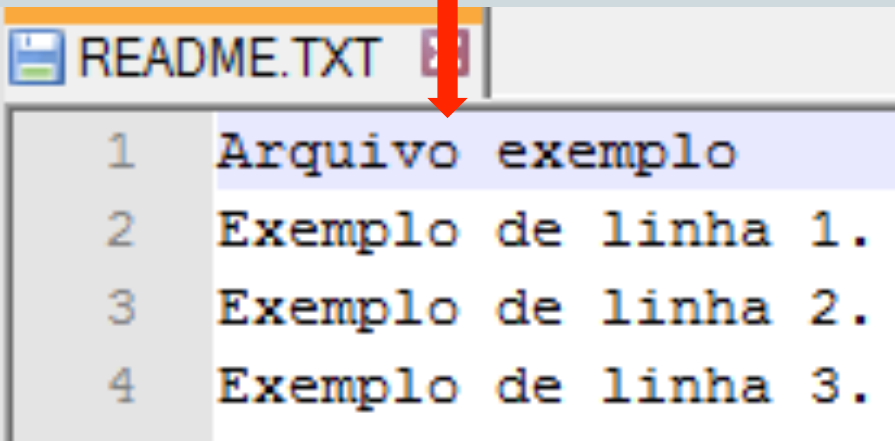
Arqu

# fgetc Passo a Passo

```
while (( ch=fgetc(fp)) != EOF)  
    putchar(ch);
```



1 Arquivo exemplo  
2 Exemplo de linha 1.  
3 Exemplo de linha 2.  
4 Exemplo de linha 3.



1 Arquivo exemplo  
2 Exemplo de linha 1.  
3 Exemplo de linha 2.  
4 Exemplo de linha 3.

Arqui

Arquiv

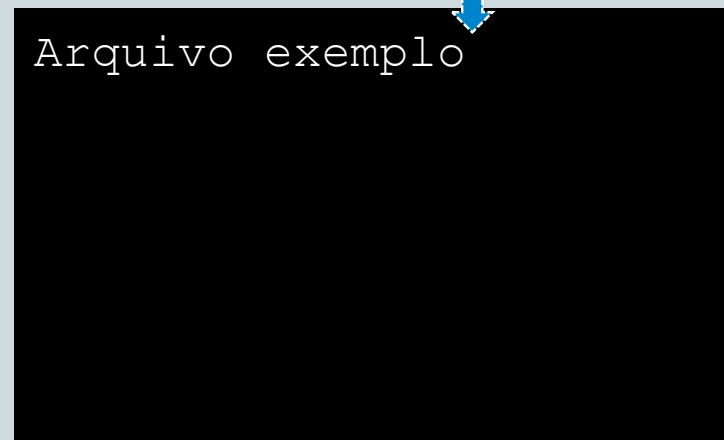
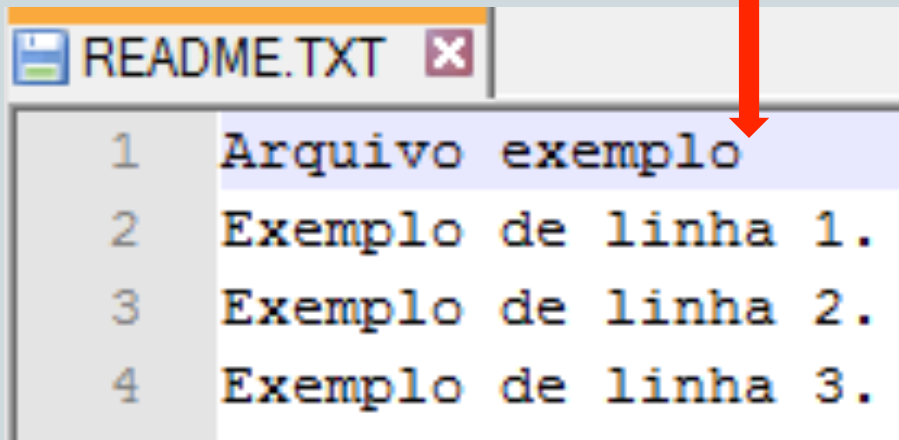


# fgetc Passo a Passo



- Ao encontrar um caractere de quebra de linha (`\n`), este é impresso normalmente.
- O cursor de leitura vai para o próximo caractere, o “E”
- O Cursor de impressão na tela é transportado para a próxima linha.

```
while (( ch=fgetc(fp)) != EOF)
    putchar(ch);
```

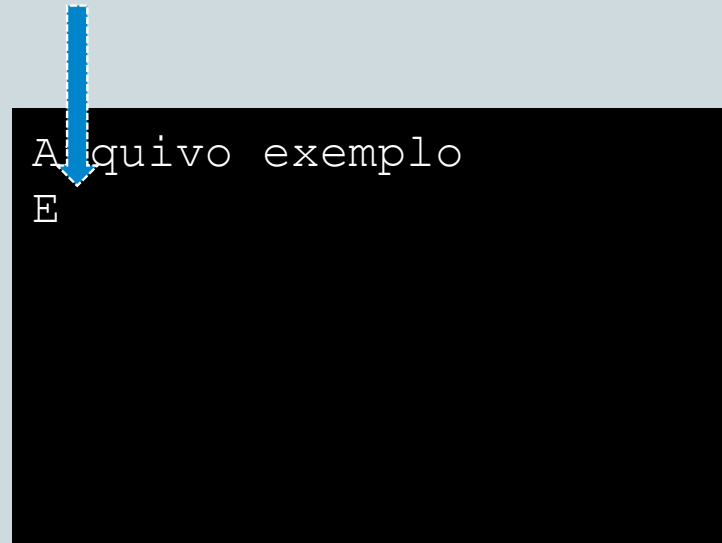
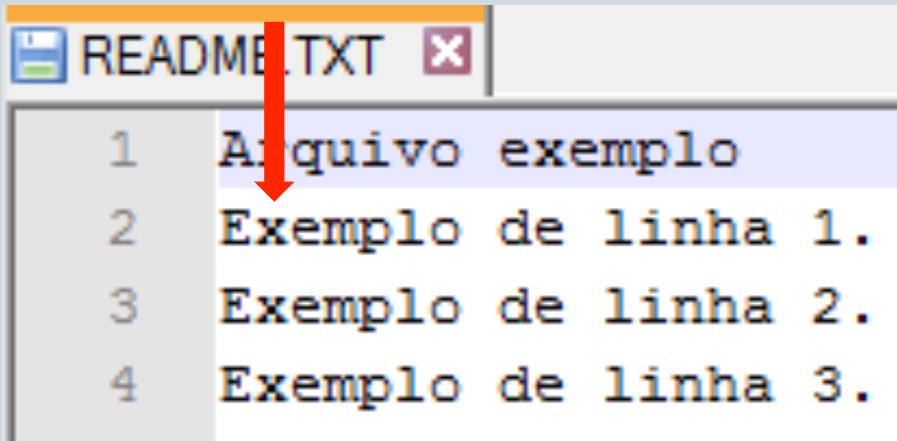


# fgetc Passo a Passo



- Após a impressão da quebra de linha, o laço **while** continua executando a leitura de caracteres com o **fgetc** e impressão com o **putchar**.

```
while (( ch=fgetc(fp)) != EOF)
    putchar(ch);
```

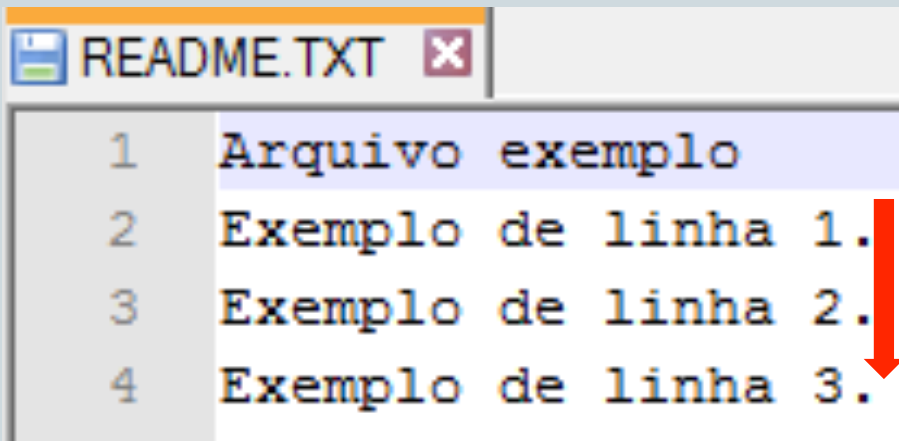


# fgetc Passo a Passo



- Ao encontrar, após a leitura e impressão do último caractere do arquivo, no exemplo, o caractere “.” não existe mais **nenhum caractere** a ser lido.
- Neste ponto o comando **fgetc** retorna **EOF** e loop de leitura termina.

```
while (( ch=fgetc(fp)) != EOF)
    putchar(ch);
```



```
Arquivo exemplo
Exemplo de linha 1.
Exemplo de linha 2.
Exemplo de linha 3.
```

# Escrita de Caracteres em um Arquivo



- Pode-se gravar um caractere no arquivo utilizando a função **fputc** (*fite put char*).

```
Int fputc (FILE *arq)
```

- A função **ESCREVE** um caractere em um arquivo aberto pelo comando **fopen**.
- Para que seja possível escrever em um arquivo usando a função **fputc** é necessário abrir o arquivo com opções de **ESCRITA**.

# Escrita de Caracteres em um Arquivo



Modo	Descrição
w	<ul style="list-style-type: none"><li>• Cria um <b>NOVO</b> arquivo vazio para <b>ESCRITA</b>.</li><li>• Caso um arquivo com o mesmo nome já existe ele será substituído por um arquivo vazio.</li></ul>
w+	<ul style="list-style-type: none"><li>• Cria um <b>NOVO</b> arquivo vazio para <b>LEITURA e ESCRITA</b>.</li><li>• Caso um arquivo com o mesmo nome já existe ele será substituído por um arquivo vazio.</li></ul>
r	<ul style="list-style-type: none"><li>• <del>Abertura de arquivo <b>EXISTENTE</b> somente para <b>LEITURA</b>.</del></li><li>• <del>Caso não possa ser aberto (não existe ou já foi aberto por outro programa) a função retorna NULL.</del></li></ul>
r+	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> para <b>LEITURA e ESCRITA</b>.</li><li>• É necessário que o arquivo exista.</li><li>• Caso não possa ser aberto a função retorna NULL.</li></ul>
a	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> para <b>ESCRITA</b> no <b>FINAL do arquivo</b>.</li><li>• Não apaga o conteúdo pré-existente no arquivo. (O "a" vem do inglês <i>append</i>, adicionar, apender)</li></ul>
a+	<ul style="list-style-type: none"><li>• Abertura de arquivo <b>EXISTENTE</b> para <b>ESCRITA</b> no <b>FINAL do arquivo e LEITURA</b>.</li></ul>

# Exemplo de Gravação de Arquivo



```
#include <stdio.h>
int main()
{
    FILE *fp;
    char ch;

    fp = fopen ("MEU_TEXTO.TXT", "w");

    if (fp == NULL) {
        printf ("Houve um erro ao criar o arquivo.\n");
    }
    else {
        printf("Digite o texto para gravar no arquivo.\n");
        printf("Digite ; para parar de gravar.\n");

        do {
            ch = getch();    // Lê Caractere do teclado
            putc(ch, stdout); // Mostra Caractere na tela
            fputc(ch, fp);   // Grava Caractere no arquivo
        }
        while (ch != ';');
        fclose (fp);
    }
    return;
}
```

```
Digite o texto para gravar no arquivo.
Digite ; para parar de gravar.
Grava texto no arquivo ate encontrar um ponto e virgula na entrada do teclado;
Process returned 0 (0x0)   execution time : 22.655 s
Press any key to continue.
-
```

MEU\_TEXTO.TXT

```
1 Grava texto no arquivo ate encontrar um ponto e virgula na entrada do teclado;
```

# Exemplo de Gravação de Arquivo



```
#include <stdio.h>
int main()
{
    FILE *arqOri, *arqDes; // Arquivos de Origem e Destino
    char ch;

    arqOri = fopen ("README.TXT", "r");
    arqDes = fopen ("MEU_TEXTO.TXT", "a");

    if (arqOri == NULL || arqDes == NULL) {
        printf ("Houve um erro ao abrir os arquivos.\n");
    }
    else {
        while ((ch = fgetc(arqOri)) != EOF){
            fputc(ch,arqDes); // Grava no arquivo de Destino
            putc(ch, stdout); // Mostra Caractere na tela
        };

        fclose(arqOri);
        fclose(arqDes);
    }
    return 0;
}
```

MEU\_TEXTO.TXT

1 Grava Texto no arquivo ate encontrar um ponto e virgula na entrada do teclado;

```
#include <stdio.h>
int main()
{
    FILE *arqOri, *arqDes; // Arquivo de Origem e Destino
    char ch;

    arqOri = fopen("README.TXT", "r");
    arqDes = fopen("MEU_TEXTO.TXT", "a");

    if (arqOri == NULL || arqDes == NULL) {
        printf("Houve um erro ao abrir os arquivos.\n");
    }
    else {
        while ((ch = fgetc(arqOri)) != EOF){
            fputc(ch, arqDes); // Grava no arquivo de Destino
            putc(ch, stdout); // Mostra Caractere na tela
        };
    }
    ret
}
```

README.TXT

1 Arquivo exemplo  
2 Exemplo de linha 1.  
3 Exemplo de linha 2.  
4 Exemplo de linha 3.

```
Arquivo exemplo
Exemplo de linha 1.
Exemplo de linha 2.
Exemplo de linha 3.
Process returned 0 (0x0)   execution time : 0.230 s
Press any key to continue.
```

EOF  
MEU\_TEXTO.TXT

MEU\_TEXTO.TXT

1 Grava Texto no arquivo ate encontrar um ponto e virgula na entrada do teclado;Arquivo exemplo  
2 Exemplo de linha 1.  
3 Exemplo de linha 2.  
4 Exemplo de linha 3.

Resultado



# Gravação de String de Texto



- **fprintf** - Para gravar uma string de texto formatada no arquivo.

```
int fprintf(FILE *arq, const char *texto, ...)
```

- **arq**: Arquivo aberto pelo **fopen**.
- **texto**: texto a ser impresso, permite formatação como no comando **printf**.
- **...** : variáveis a serem substituídas

# Exemplo de Gravação de String de Texto

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

```
int main(void)
{
```

```
    char url[]="notas.txt";
    float nota,
          media=0.0;
    FILE *arq;
```

```
    arq = fopen(url, "w");
    if(arq == NULL)
```

```
        printf("Erro, nao foi possivel abrir o arquivo\n");
```

```
    else{
```

```
        printf("M1: ");
```

```
        scanf("%f", &nota);
```

```
        fprintf(arq, "M1: %.2f\n", nota);
```

```
        media+=nota;
```

```
        printf("M2: ");
```

```
        scanf("%f", &nota);
```

```
        fprintf(arq, "M2: %.2f\n", nota);
```

```
        media+=nota;
```

```
        media /= 2;
```

```
        fprintf(arq, "Media: %.2f\n", media);
```

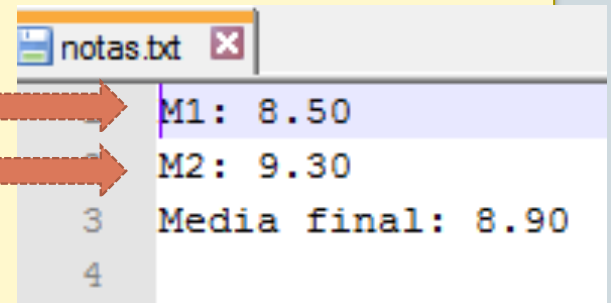
```
    }
```

```
    fclose(arq);
```

```
    return 0;
```

```
M1: 8.5
M2: 9.3
```

```
Process returned 0 (0x0)   execution time : 7.617 s
Press any key to continue.
```



```
notas.txt
M1: 8.50
M2: 9.30
Media final: 8.90
```

# Leitura de String de Texto



- **fscanf** - Para ler uma string de texto formatada no arquivo.

```
int fscanf(FILE *arq, const char *texto, ...)
```

- **arq**: Arquivo aberto pelo **fopen**.
- **texto**: texto a ser lido, permite formatação como no comando **scanf**.
- **...** : variáveis que receberão o conteúdo do arquivo

# Exemplo de Leitura de String de Texto

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

```
int main(void)
```

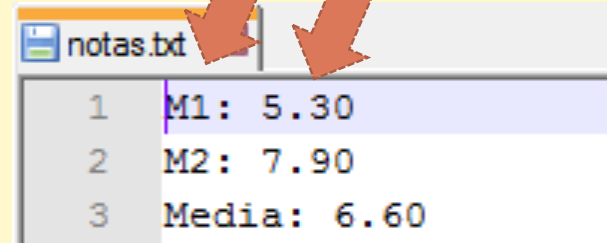
```
{
    char url[]="notas.txt";
    float nota;
    char str[10];
    FILE *arq;
```

```
    arq = fopen(url, "r");
    if(arq == NULL)
        printf("Erro, nao foi possivel abrir o arquivo\n");
    else{
        // Lê uma string e um float
        while (fscanf(arq, "%s %f", str, &nota) > 0){
            // Mostra valores lidos na tela
            printf("%s %.2f\n", str, nota);
        }
    }
```

```
    fclose(arq);
```

```
    return 0;
```

```
}
```



```
M1: 5.30
M2: 7.90
Media: 6.60
```

```
Process returned 0 (0x0)   execution time : 0.230 s
Press any key to continue.
```

# Leitura e Escrita em Arquivos Binários



- Funções **fgetc** e **fputc**: utilizadas para **ler** e **gravar** em **arquivos texto**.
- Funções **fread** e **fwrite**: utilizadas para **ler** e **gravar** em arquivos **TEXTO** ou **BINÁRIOS**.

```
int fwrite(const void *prt, int size, int n, FILE *arq)
```

```
int fread(const void *prt, int size, int n, FILE *arq)
```

# Escrita em Arquivos Binários



```
int fwrite(const void *prt, int size, int n, FILE *arq)
```

- **prt**: ponteiro para void (qualquer tipo), contendo o endereço de memória daquilo que queremos escrever no arquivo.
- **size** : tamanho em bytes de cada elemento a ser escrito.
- **n**: número de elementos a serem escritos.
- **arq**: arquivo, aberto pelo **fopen**, onde os registros serão gravados.
- **retorno**: número de itens que foi possível escrever no arquivo.

# Exemplo de Escrita em Arquivos Binários

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

```
struct CLIENTE{
    int cod;
    char nome[30];
}
```

Arquivo binário, quando aberto em um editor de texto.

```
int main()
{
    FILE *arqDados;
    struct CLIENTE cli;

    arqDados = fopen ("DADOS.DAT", "wb");
```

DADOS.DAT x

SOHNULNULNULElianaNULv^SYúpyybDC1·vÄ[·wDEM@NUL"ÿ(NULSTXNULNULNULClaudioNUL^SYúpyybDC1·vÄ[·wDEM@NUL"ÿ(NUL

```
cli.cod = 1;
strcpy(cli.nome, "Eliana");
fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);

// Insere Cliente 2
cli.cod = 2;
strcpy(cli.nome, "Claudio");
fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);

fclose(arqDados);
```

```
}
return 0;
}
```

# Exemplo de Escrita em Arquivos Binários

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

```
struct CLIENTE{
    int cod;
    char nome[30];
};
```

## Estrutura

Um tipo de dado não definido inicialmente. Não é int, não é char...

Necessidade SÓ MINHA.

```
int main()
{
```

```
FILE *arqDados;
struct CLIENTE cli;
```

Variável do “meu tipo” struct.

```
arqDados = fopen ("DADOS.DAT", "wb");
```

**wb** indica escrita (write), binária.

```
if (arqDados == NULL) {
    printf ("Houve um erro ao criar o arquivo.\n");
} else {
```

```
    // Insere Cliente 1
```

```
    cli.cod = 1;
    strcpy(cli.nome, "Eliana");
```

Meu Cliente

```
    fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);
```

```
    // Insere Cliente 2
```

```
    cli.cod = 2;
    strcpy(cli.nome, "Claudio");
```

Meu Cliente

```
    fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);
```

```
    fclose(arqDados);
```

Finalmente

o **fwrite** escreve o dado no arquivo

```
    }
    return 0;
}
```



# Exemplo de Escrita em Arquivos Binários

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

```
struct CLIENTE{
    int cod;
    char nome[30];
};
```

```
int main()
{
```

```
    FILE *arqDados;
```

```
    struct CLIENTE cli;
```

Variável do “meu tipo” struct.

```
    arqDados = fopen ("DADO
```

Contém o conteúdo que será escrito no arquivo

```
    if (arqDados == NULL) {
```

```
        printf ("Houve um erro ao criar o arquivo.\n");
```

```
    } else {
```

```
        // Insere Cliente 1
```

```
        cli.cod = 1;
```

```
        strcpy(cli.nome, "Eliana");
```

```
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);
```

```
        // Insere Cliente 2
```

```
        cli.cod = 2;
```

```
        strcpy(cli.nome, "Claudio");
```

```
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);
```

```
        fclose(arqDados);
```

```
    }
```

```
    return 0;
```

```
}
```

O “&” extrai o endereço da variável **cli** que contém os dados a serem escritos.

# Exemplo de Escrita em Arquivos Binários

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

```
struct CLIENTE{
    int cod;
    char nome[30];
};
```

## Estrutura

Um tipo de dado não definido inicialmente. Não é int, não é char...

Necessidade SÓ MINHA.

```
int main()
{
```

```
    FILE *arqDados;
    struct CLIENTE cli;
```

```
    arqDados = fopen ("DADOS.DAT", "wb");
```

```
    if (arqDados == NULL) {
        printf ("Houve um erro ao criar o arquivo.\n");
    } else {
```

```
        // Insere Cliente 1
        cli.cod = 1;
        strcpy(cli.nome, "Eliana");
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);
```

```
        // Insere Cliente 2
        cli.cod = 2;
        strcpy(cli.nome, "Claudio");
```

```
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);
```

```
        fclose(arqDados);
```

```
    }
    return 0;
}
```

Tamanho, em bytes, do que está sendo escrito.

A função sizeof, retorna o tamanho da estrutura Cliente

Arquivo de destino

O **1** indica que será escrita 1 registro do tamanho indicado pelo **sizeof**. Poderia ser mais de um, no caso de uma **fila** ou **lista**, por exemplo.

# Exemplo de Leitura de Arquivos Binários

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

```
struct CLIENTE{
    int cod;
    char nome[30];
};
```

## Estrutura a ser lida

Um tipo de dado não definido inicialmente. Não é int, não é char...

Deve ser igual à que foi gravada

```
int main()
{
```

```
    FILE *arqDados;
    struct CLIENTE cli;
```

Variável do “meu tipo” struct.

```
    arqDados = fopen ("DADOS.DAT", "rb");
```

**rb** indica leitura (**read**), **binária**.

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

```
    else {
        printf("==== Leitura do arquivo DADOS.DAT ==== \n");
        while ((fread(&cli, sizeof(struct CLIENTE), 1, arqDados) > 0))
```

```
            printf("Codigo: %d; Nome: %s;\n", cli.cod, cli.nome);
```

```
    }
```

```
    }
    return 0;
```

```
}
```

O laço **while** irá repetir enquanto o **fread** conseguir ler algo, ou seja, retornar > zero.

Quando **fread** retorna zero, é porque não há nada a ser lido, ou seja, fim que arquivo (**EOF**)

# Arquivos de Entrada e Saída Padrão



- Arquivo (stream) **stdout**: Saída Padrão → **Tela**
- Arquivo (stream) **stdin**: Entrada Padrão → **Teclado**

Comando	Similar usando Arquivo (stream)
<code>printf("Um dois Três");</code>	<code>fprintf(stdout, "Um dois Três");</code>
<code>puts("Ola");</code>	<code>fputs("Ola\n", stdout);</code>
<code>scanf("%d %c", &amp;a, &amp;b);</code>	<code>fscanf(stdin, "%d %c", &amp;a, &amp;b);</code>

# Acesso Sequencial x Acesso Direto



- Ao abrir um arquivo, o comando **fopen** posiciona-se o cursor no primeiro byte do arquivo. Posição Zero.
- Até agora: **Acesso Sequencial**. Percorre-se byte a byte do arquivo, desde o primeiro.

```
while (( ch=fgetc(fp)) != EOF)
    putchar(ch);
```

```
while ((fread(&cli, sizeof(struct CLIENTE), 1, arqDados)) > 0){
    printf("Codigo: %d; Nome: %s;\n", cli.cod, cli.nome);
}
```

- É possível saber em que posição encontra-se o cursor no arquivo com o comando **ftell**.

```
long ftell(FILE *arq)
```

- É possível voltar o cursor à posição Zero, utilizando o comando **rewind**.

```
void rewind(FILE *arq)
```

# Acesso Sequencial x Acesso Direto



- **Acesso Direto:** Posiciona-se o cursor em um **ponto específico do arquivo**.
- Não é necessário percorrer byte a byte.

```
int fseek(FILE *arq, long salto, int origem)
```

- **arq:** Arquivo onde o cursor será posicionado
- **salto:** Número de bytes que o cursor deve andar.
- **origem:** Local a partir do qual deseja-se realizar o salto. Pode-se utilizar 3 valores, definidos pelas constantes.

Constante	Valor	Significado
SEEK_SET	0	O salto é realizado a partir da origem do arquivo.
SEEK_CUR	1	O salto é realizado a partir da posição corrente do arquivo.
SEEK_END	2	O salto é realizado a partir do final do arquivo.

# Exemplo de Acesso Direto

- Suponha o arquivo com 3 registros.
- Note o crédito de cada cliente

```
#include <stdio.h>
#include <string.h>
struct CLIENTE{
    int cod;
    char nome[30];
    float credito;
};
int main()
{
    FILE *arqDados; //
    struct CLIENTE cli;
    arqDados = fopen ("CLIENTES.DAT", "wb");
    if (arqDados == NULL) {
        printf ("Houve um erro ao criar o arquivo.\n");
    }
    else {
        // Insere Cliente 1
        cli.cod = 1;
        strcpy(cli.nome, "Eliana");
        cli.credito = 1500.00;
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);

        // Insere Cliente 2
        cli.cod = 2;
        strcpy(cli.nome, "Claudio");
        cli.credito = 5500.00;
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);

        // Insere Cliente 3
        cli.cod = 3;
        strcpy(cli.nome, "Antonio");
        cli.credito = 1000.00;
        fwrite(&cli, sizeof(struct CLIENTE), 1, arqDados);

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

# Exemplo de Acesso Direto

- Programa que Busca pelo Maior Crédito

Variáveis para Identificar o cliente com maior crédito.

Busca Sequencial para identificar cliente com maior crédito.

Após Identificado por **PosMaiorCredito**, o faz um **Acesso Direto** ao registro para pegar os dados

```
#include <stdio.h>
#include <string.h>
struct CLIENTE{
    int cod;
    char nome[30];
    float credito;
```

```
int main()
{
    FILE *arqDados;
    struct CLIENTE cli;

    float maiorCredito = 0;
    long PosMaiorCredito;

    arqDados = fopen ("CLIENTES.DAT", "rb");

    if (arqDados == NULL) {
        printf ("Houve um erro ao abrir o arquivo.\n");
    }
    else {
        // Acesso sequencial, para descobrir o maior Crédito.
        while ((fread(&cli, sizeof(struct CLIENTE), 1, arqDados)) > 0){
            if (cli.credito > maiorCredito){
                maiorCredito = cli.credito;
                // Pega a posição do maior Crédito
                PosMaiorCredito = ftell(arqDados) - sizeof(struct CLIENTE);
            }
        }
        // Acesso Direto para mostrar o registro com o maior Crédito
        fseek(arqDados, PosMaiorCredito, SEEK_SET);
        fread(&cli, sizeof(struct CLIENTE), 1, arqDados);

        printf("==== Cliente com Maior Crédito ==== \n");
        // Mostra os dados do registro com o maior Crédito
        printf("Codigo: %d; Nome: %s; Credito: %f ;\n", cli.cod, cli.nome, cli.credito);
    }
}
```

```
==== Cliente com Maior Crédito ====
Codigo: 3; Nome: Antonio; Credito: 5500.000000 ;
Process returned 0 (0x0)   execution time : 0.220 s
Press any key to continue.
```



# Resumo de funções

fopen

- **Abre** ou **Cria** um arquivo texto ou binário
- Define operação de **Leitura** e/ou **Gravação**

Posição

- ftell → **Posição atual** do cursor no arquivo.
- fseek → Movimenta o cursor diretamente para **posição específica** do arquivo.
- rewind → Volta para o **início** do arquivo.

Leitura e/  
ou Escrita

- **Arquivo TEXTO** → fgets e fputs OU fread e fwrite, além do fscanf e fprintf.
- **Arquivo BINÁRIO** → fread e fwrite.

fflush

- Grava os dados em memória para o arquivo.
- Mantém o arquivo aberto para leitura e/ou gravação.
- Execução **opcional**.

fclose

- Grava os dados em memória para o arquivo.
- Fecha o arquivo.
- Nenhuma leitura ou gravação é permitida sem que o arquivo seja reaberto.