

# LINGUAGEM DE PROGRAMAÇÃO II

---

© 2022, Gizelle Kupac Vianna (DECOMP/UFRRJ)

# ARQUIVOS

---

## Aula 6

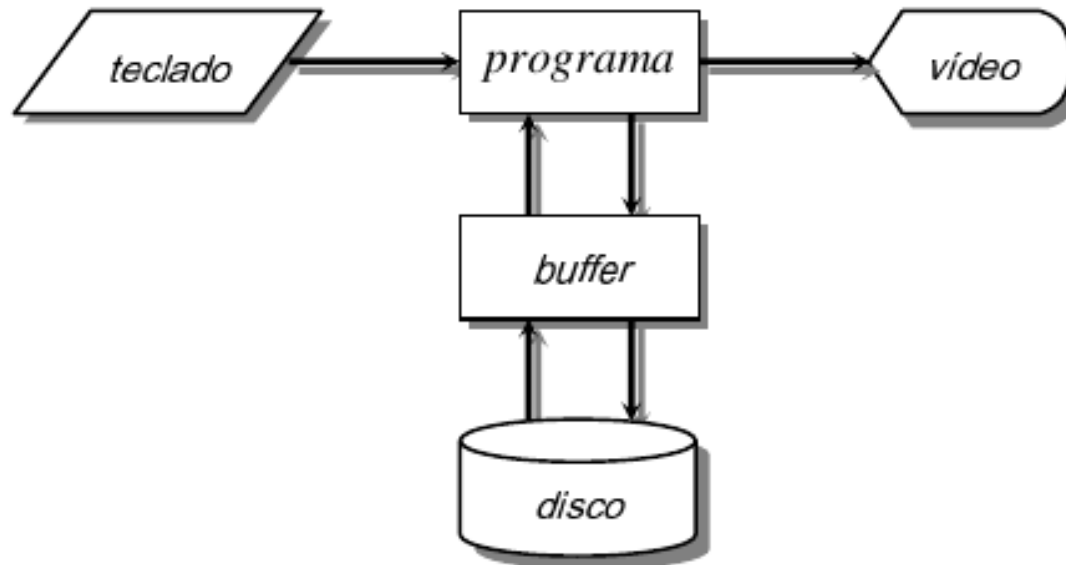
# Arquivos

- Um arquivo é uma coleção homogênea de itens armazenados em meio físico.
- A utilização de arquivos permite que os dados não sejam perdidos entre uma execução e outra. Com eles, podemos armazenar as informações geradas pelo programas no disco rígido, para consulta posterior.
- O acesso a disco é muito mais lento do que o acesso à memória e o uso de arquivos impacta no tempo de execução do programa. Para resolver esse problema, existe uma solução transparente ao programador.

# Buffer

- Para melhorar a eficiência, o sistema operacional usa uma área de memória denominada **buffer**;
- Os dados gravados pelo programa são temporariamente armazenados no buffer. Quando ele fica cheio, o sistema o descarrega de uma só vez no disco.
- Durante a leitura, o sistema se encarrega de encher o buffer toda vez que ele fica vazio; Isso diminui o número de acessos a disco e, portanto, aumenta a velocidade de execução do programa.

# Buffer



# Sintaxe

- Para se trabalhar com arquivos é preciso primeiro criar variáveis do tipo arquivo:
  - `FILE *arq, *entrada, *saida;`
- Podemos criar diversas variáveis de arquivo, cada uma delas devendo ser associada a um arquivo distinto no disco;

# Operações Básicas

- Abertura de arquivo:
  - Essa operação estabelece a conexão entre o programa na memória e o arquivo em disco.
  - Durante a abertura de arquivo são alocados os espaços para armazenamento da estrutura do tipo FILE e o buffer de transferência de dados.
- Sintaxe:
  - *a = fopen ("nome", "modo");*

# Modos de Abertura

- Modo Texto:
  - O arquivo é visto como uma sequência de caracteres. Pode ser lido por uma pessoa e editado em um editor de textos.
  - Quando aberto em modo texto, todo numeral que precisar ser gravado no arquivo será antes convertido para uma string.
  - Durante a leitura, o sistema irá fazer o oposto e converter strings em numerais, segundo as instruções do código.
- Modo Binário:
  - O arquivo é visto como uma sequência de bytes.
  - os números são armazenados em disco sem que nenhuma conversão precise ser feita.
  - Permite salvar e recuperar grandes quantidades de dados de forma mais eficiente.




# Operações Básicas

- Durante a tentativa de abertura de um arquivo, podem ocorrer alguns erros. Por exemplo, tentar abrir um arquivo que não existe, porque digitamos seu nome errado.
- Dependendo do sistema operacional sendo usado, tentativas de abertura de arquivos do sistema não são permitidas, como no Linux.
- Caso você tente alterar ou acessar um arquivo sem permissão, ou inexistente, a função *fopen* irá retornar NULL para o ponteiro.

# Exemplo

```
FILE *e;
```

```
e = fopen("entrada.dat", "r");  ABERTO PARA LEITURA!  
if( e==NULL ) {  
    printf("\nArquivo não pode ser aberto");  
    exit(1);  
}
```

# Exemplo

```
FILE *s;
```

```
if ((s=fopen("saida.dat", "w")) == NULL) ; {
```



**ABERTO PARA ESCRITA!**

```
    printf("\nArquivo não pode ser aberto");
```

```
    exit(1);
```

```
}
```

# Exemplo

```
FILE *f;
```

```
if( (f=fopen("saida.dat", "a")) == NULL ) {
```



**ABERTO PARA ACRÉSCIMO!**

```
    printf("\nArquivo não pode ser aberto");
```

```
    exit(1);
```

```
}
```

# Operações Básicas

- Fechamento de arquivo:
  - Essa operação esvazia o buffer, salvando os dados que ainda restam nele em disco.
  - Após o fechamento do arquivo, os espaços para armazenamento da estrutura do tipo FILE e o buffer de transferência de dados são liberados e devolvidos à memória.
  - O fechamento é importante para evitar inconsistências.
- Sintaxe:
  - *fclose (a)*, ou *fcloseall()*

# Exemplo

```
FILE *a, *e, *s;
```

```
...
```

```
fclose(s); /* fecha somente o arquivo s */
```

```
fcloseall(); /* fecha os arquivos restantes */
```

```
...
```

# Modos de Abertura

Modo	Significado
"r"	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
"w"	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
"a"	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"rb"	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
"wb"	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
"ab"	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.
"r+"	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
"w+"	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
"a+"	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"r+b"	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
"w+b"	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
"a+b"	Acrescenta dados ou cria um arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

# Leitura e Escrita em Arquivos

- De modo muito similar aos comandos de leitura e escrita de dados no terminal, podemos ler e escrever dados nos arquivos.
- Existem algumas funções pré-definidas que executam a tarefa de salvar informações nos arquivos. Chamamos essa operação de “escrita” em arquivos.
- A operação oposta seria a recuperação de dados armazenados em um arquivo, que chamamos de “leitura” em arquivos.



# Verificação de final de arquivo

- Durante a leitura de um arquivo, frequentemente precisamos saber se todos os seus dados já foram lidos.
- Para isso, usamos a função *feof()* , que informa quando o final de arquivo foi atingido.
- Existe também uma constante pré-definida, a **EOF**, que também pode ser usada para verificar o final do arquivo.

# ARQUIVOS EM MODO TEXTO

---

# Escrita em Arquivos

- *int **fputc**(int c, FILE \*arq):*
  - A função recebe o caractere a ser escrito e o ponteiro para o arquivo alvo.
  - Ela retorna o caractere *EOF* caso não tenha conseguido escrever no arquivo, ou o inteiro que representa o caractere, caso tenha sucesso.

# Exemplo 1 – Escrita em Modo Texto

```
#include <stdio.h>

main() {
    char letra;
    int cod;
    FILE *f;

    printf("Digite uma letra qualquer: ");
    scanf("%c", &letra);

    f = fopen("letra.txt", "w");
    if (f == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else{
        cod = fputc(letra, f);
        printf("\nSalvamos a letra %d no arquivo",cod);
        fclose(f);
    }
    return 0;
}
```

# Escrita em Arquivos

- *int **fprintf**(FILE \*arq, char \*formato, ...) :*
  - Permite a escrita de strings inteiras em um único comando.
  - É similar à *printf()*, com um parâmetro adicional, que indica para o arquivo onde o dado será salvo.
  - Retorna a quantidade de bytes escritos no arquivo, mas seu uso é opcional.
  - Ela retorna o caractere *EOF* caso não tenha conseguido escrever no arquivo.

# Exemplo

```
main() {
    char materia[31];
    float nota;
    FILE *f;

    printf("Digite o nome da disciplina: ");
    gets(materia);
    printf("e a nota: ");
    scanf("%f", &nota);

    f = fopen("letra.txt", "w");
    if (f == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else{
        fprintf(f,"%s %f \n", materia, nota);
        fclose(f);
    }
    return 0;
}
```

# Leitura em Arquivos

- *int fscanf(FILE \*arq, char \*formato, ...)* :
  - Permite a leitura de strings inteiras em um único comando.
  - É similar à *scanf()*, com um parâmetro adicional que indica para o arquivo onde o dado será salvo.
  - A cada leitura, os dados são transferidos para a memória e o cursor do arquivo avança, passando para o próximo dado do arquivo.
  - Retorno a quantidade de bytes lidos com sucesso, ou *EOF* quando o final do arquivo for alcançado.

# Exemplo

```
#include <stdio.h>

main() {
    char linha[30];
    int nlinhas = 0;
    float nota;
    FILE *f;

    f = fopen("notas.txt", "r");
    if (f == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else {
        while(!feof(f)) {
            fscanf(f, "%s %f", linha, &nota);
            printf("\n%s %.2f", linha, nota);
        }
    }
    fclose(f);
    return 0;
}
```



# Leitura em Arquivos

- *int **fgetc** (FILE \*arq) :*
  - Permite a leitura de caracteres e retorna o código ASCII do caractere lido.
- *char\* **fgets** (char \* s, int n, FILE \*arq) :*
  - Lê uma sequência de caracteres, até encontrar um '\n' ou atingir o máximo de caracteres *n-1*, pois é acrescentado o caractere '\0' ao final da string s.
  - Retorna o código ASCII do caractere lido.
  - O valor de retorno é o endereço da string s.

# Exemplo

```
#include <stdio.h>

main() {
    char letra;
    int nlinhas = 0;
    FILE *f;

    f = fopen("notas.txt", "r");
    if (f == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else{
        while ((letra = fgetc(f)) != EOF) {
            if (letra == '\n')
                nlinhas++;
        }
        printf("\nQuantidade de linhas = %d",nlinhas);
        fclose(f);
    }

    return 0;
}
```

# Exemplo

```
##include <stdio.h>

main() {
    char linha[30];
    int nlinhas = 0;
    FILE *f;

    f = fopen("notas.txt", "r");
    if (f == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else {
        while ( fgets(linha, 30, f) != NULL )
            nlinhas++;

        printf("\nQuantidade de linhas = %d",nlinhas);
        fclose(f);
    }
    return 0;
}
```

# Leitura em Arquivos

- A informação lida é sempre a partir da posição atual do cursor do arquivo.
- Ao abrir um arquivo para leitura, esse cursor é sempre posicionado, automaticamente, no início do arquivo.
- A cada leitura, o cursor avança e passa para a próxima posição.

# Exemplo – Modo Texto

```
void inserir_registros (FILE *s) {  
    Func f;  
  
    printf("Entre com os dados de cada funcionario, digitando  
    fim para  
        finalizar o cadastramento:\n");  
    printf("\n Nome? ");  
    scanf("%s", f.nome);  
  
    while ( strcmp(f.nome,"fim") ) {  
        printf("\n Salario? ");  
        scanf("%f", &f.salario);  
        fprintf(s,"%s %f \n", f.nome, f.salario);  
        printf("\n Nome? ");  
        scanf("%s", f.nome);  
    }  
}
```

# Exemplo – Modo Texto

```
#include <stdio.h>

typedef struct {
    char nome[31];
    float salario;
} Func;

FILE * abrir_arquivo (char * nome) {
    FILE * s;

    if( (s=fopen(nome, "a")) == NULL ) {
        printf("Arquivo não pode ser aberto\n");
        return NULL;
    }
    return s;
}
```

# Exemplo – Modo Texto

```
int listar_registros (char * arq) {  
    Func f;  
    FILE * s;  
  
    if( (s=fopen(arq,"r")) == NULL ) return 0;  
  
    printf("          NOME      \t\t| \t\tSALARIO      \n");  
  
    while(!feof(s)) {  
        fscanf(s,"%s %f", f.nome, &f.salario);  
        printf("%20s\t\t\t\t%10.2f\n", f.nome, f.salario);  
    }  
    fclose(s);  
    return 1;  
}
```

# Exemplo – Modo Texto

```
main() {  
    FILE *s;  
    Func f;  
  
    listar_registros ("agenda.dat");  
    s = abrir_arquivo ("agenda.dat");  
    inserir_registros (s);  
  
    fclose(s);  
    listar_registros ("agenda.dat");  
}
```



# ARQUIVOS EM MODO BINÁRIO

---

# Escrita em Arquivos

- *int **fwrite** (tipo \* p, int tam, int qtd, FILE \*arq):*
  - O primeiro parâmetro aponta para o endereço do dado que se quer escrever no arquivo.
  - *tam* indica tamanho, em bytes, de cada elemento a ser escrito.
  - *qtd* indica a quantidade de elementos que serão escritos
  - *arq* é o ponteiro para o arquivo alvo.
  - Ela retorna NULL caso não consiga escrever no arquivo.

# Exemplo

```
typedef struct {
    int x, y, z;
} Ponto;

main() {
    Ponto p;
    FILE *f;
    int i;

    f = fopen("pontos.dat", "wb");
    if (f == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else {
        for (i=1; i<=3; i++){
            printf("\n%d ponto:\n", i);
            scanf("%d %d %d", &p.x, &p.y, &p.z);
            fwrite (&p, sizeof(Ponto), 1, f);
        }
        fclose(f);
    }
    return 0;
}
```

# Leitura de Arquivos

- *int **fread** (tipo \* p, int tam, int qtd, FILE \*arq):*
  - Similar à função *fwrite()*, a diferença é que agora o primeiro parâmetro aponta para o endereço do dado que vai receber o conteúdo extraído do arquivo.
  - Retorna NULL caso não consiga ler do arquivo.

# Exemplo

```
typedef struct {
    int x, y, z;
} Ponto;

main() {
    Ponto q;
    FILE *fp;
    int i;

    fp = fopen("pontos.dat", "rb");
    if (fp == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else {
        for (i=1; i<=3; i++){
            printf("\n%d ponto:\n", i);
            fread(&q, sizeof(Ponto), 1, fp);
            printf("Ponto %d: x = %d, y = %d, z = %d", i, q.x, q.y, q.z);
        }
    }
    fclose(fp);
    return 0;
}
```

# Reposicionando o Cursor

- *int **fseek** (FILE \*arq, long offset, int origem):*
  - Em arquivos binários podemos alterar a posição do cursor do arquivo e posicioná-lo para ler um dado específico.
  - *offset* indica quantos bytes iremos avançar.
  - *origem* indica a posição a partir de onde iremos avançar. Ela pode ser:
    - SEEK\_CUR: a posição corrente.
    - SEEK\_SET: o início do arquivo.
    - SEEK\_END: o final do arquivo.

# Exemplo

```
typedef struct {
    int x, y, z;
} Ponto;

main() {
    Ponto q;
    FILE *fp;
    int i;

    fp = fopen("pontos.dat", "rb");
    if (fp == NULL)
        printf("\nErro, nao foi possivel abrir o arquivo!");
    else {
        printf("\nQual ponto voce quer rever?");
        scanf("%d", &i);
        i--;
        fseek (fp, i*sizeof(Ponto), SEEK_SET);
        fread(&q, sizeof(Ponto), 1, fp);
        printf("Ponto %d: x = %d, y = %d, z = %d", i, q.x, q.y, q.z);
    }
    fclose(fp);
    return 0;
}
```

- Resumo das Funções

Função	Ação
<code>fopen()</code>	Abre um arquivo
<code>Fclose()</code>	Fecha um arquivo
<code>fseek()</code>	Posiciona em um registro de um arquivo
<code>fprintf()</code>	Efetua impressão formatada em um arquivo
<code>fscanf()</code>	Efetua leitura formatada em um arquivo
<code>feof()</code>	Verifica o final de um arquivo
<code>fwrite()</code>	Escreve tipos maiores que 1 byte em um arquivo
<code>fread()</code>	Lê tipos maiores que 1 byte de um arquivo