

ALGORITMOS E PROGRAMAÇÃO II

ARQUIVOS

Prof. Viviane Bonadia dos Santos

O MODELO DE FLUXO DE DADOS

"Os programas escritos em C utilizam o conceito de fluxo de dados (streams) para comunicarem-se com dispositivos do computador ou com outros programas."

"Este modelo é suficientemente poderoso para descrever com simplicidade a maioria dos dispositivos de envio ou recebimento de dados. Entre eles: teclado, terminal, arquivos, conexões de rede, comunicação entre programas, compressão de dados, entre outros."

O MODELO DE FLUXO DE DADOS

"Um fluxo de dados é a passagem de dados entre o programa e outra entidade (que pode ser um dispositivo do computador ou um outro programa)."

Arquivos: Fluxo de bytes (stream)



ARQUIVOS

- Podem armazenar **grande quantidade de informação**;
- Dados são **persistentes** (gravados em disco);
- Acesso aos dados pode ser não sequencial;
- Acesso à informação pode ser concorrente (mais de um programa ao mesmo tempo).

ARQUIVOS - NOMES E EXTENSÕES

- Arquivos são identificados por um nome;
- O nome de um arquivo pode conter uma extensão que indica o tipo de conteúdo do arquivo.

arq.txt	Arquivo texto simples
arq.c	Código-fonte em C
arq.pdf	<i>Portable document format</i>
arq.html	Arquivo para páginas WWW
...	...
arq.exe	Arquivo executável (Windows)

TIPOS DE ARQUIVOS

- Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:
 - **Arquivo texto:** Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples
 - **Exemplos:** código-fonte C, documento texto simples, páginas HTML

TIPOS DE ARQUIVOS

- Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:
 - **Arquivo binário:** Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente
 - **Exemplos:** arquivos executáveis, arquivos compactados, documentos do Word

ABRIR E FECHAR FLUXOS DE DADOS

Antes de realizar leituras ou escritas, é necessário abrir o fluxo. Esta operação tem vários propósitos:


Associa (ou conecta) o fluxo com o dispositivo desejado.



ABRIR E FECHAR FLUXOS DE DADOS

Antes de realizar leituras ou escritas, é necessário abrir o fluxo. Esta operação tem vários propósitos:

Associa (ou conecta) o fluxo com o dispositivo desejado.

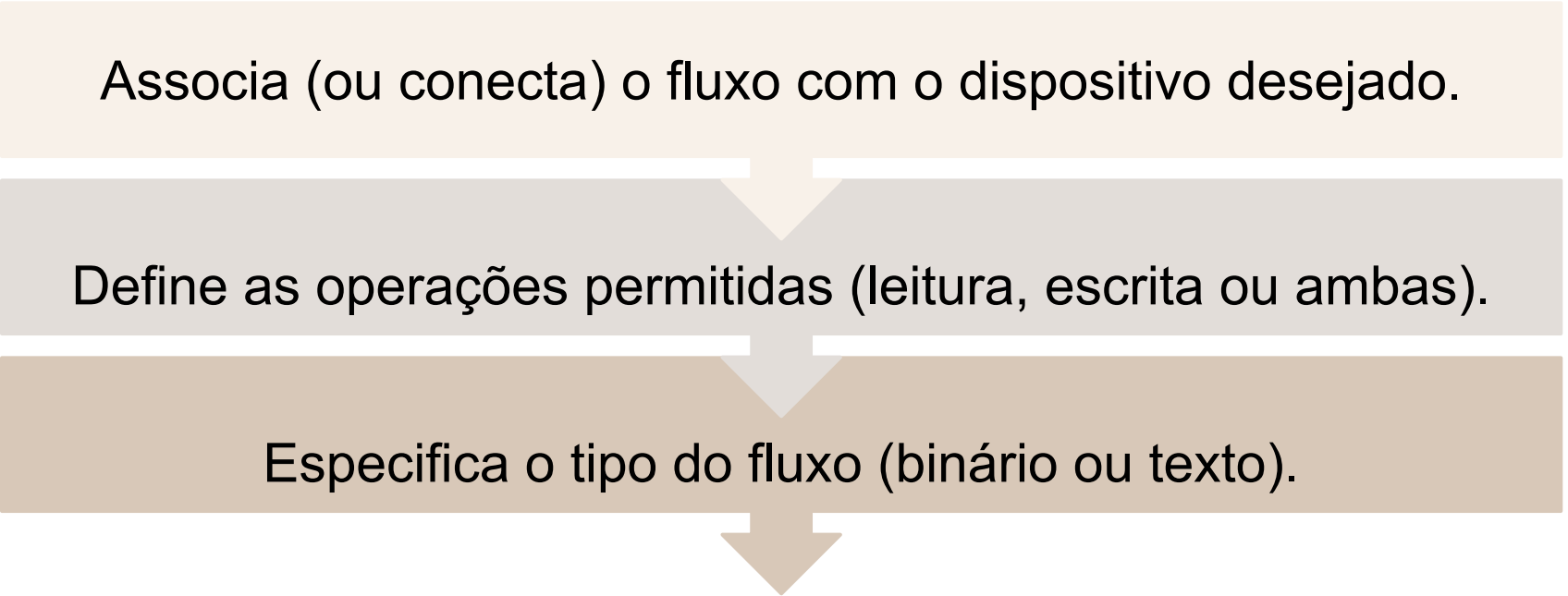


Define as operações permitidas (leitura, escrita ou ambas).



ABRIR E FECHAR FLUXOS DE DADOS

Antes de realizar leituras ou escritas, é necessário abrir o fluxo. Esta operação tem vários propósitos:



```
graph TD; A[Associa (ou conecta) o fluxo com o dispositivo desejado.] --> B[Define as operações permitidas (leitura, escrita ou ambas).]; B --> C[Especifica o tipo do fluxo (binário ou texto).]; C --> D[ ];
```

Associa (ou conecta) o fluxo com o dispositivo desejado.

Define as operações permitidas (leitura, escrita ou ambas).

Especifica o tipo do fluxo (binário ou texto).

ABRIR E FECHAR FLUXOS DE DADOS

Antes de realizar leituras ou escritas, é necessário abrir o fluxo. Esta operação tem vários propósitos:

Associa (ou conecta) o fluxo com o dispositivo desejado.

Define as operações permitidas (leitura, escrita ou ambas).

Especifica o tipo do fluxo (binário ou texto).

Escolhe entre acesso sequencial ou aleatório (se disponível).

ABRIR E FECHAR FLUXOS DE DADOS

Antes de realizar leituras ou escritas, é necessário abrir o fluxo. Esta operação tem vários propósitos:

Associa (ou conecta) o fluxo com o dispositivo desejado.

Define as operações permitidas (leitura, escrita ou ambas).

Especifica o tipo do fluxo (binário ou texto).

Escolhe entre acesso sequencial ou aleatório (se disponível).

Fecha o fluxo.

ABRIR E FECHAR UM ARQUIVO

- Em C, um fluxo de dados associado com arquivos é declarado como uma variável de tipo **FILE ***
- Arquivos são manipulados como variáveis do tipo apontador para arquivo, as quais são declaradas da seguinte forma:

```
int main () {  
    FILE *arq;  
    ...  
    return 0;  
}
```

As funções necessárias para manipulação estão na biblioteca `stdio.h`

ABRIR E FECHAR UM ARQUIVO

- Antes de realizar leitura ou escrita em um arquivo, é necessário abri-lo usando a função **fopen**.

```
int main () {  
    FILE *arq;  
    arq = fopen("nome do arquivo", "modo");  
    ...  
    fclose(arq);  
    return 0;  
}
```

ABRIR E FECHAR UM ARQUIVO

- O comando **fopen** possui dois argumentos: nome do arquivo e modo. Se fopen falhar, então o valor da variável **arq** será **NULL**.

```
int main () {  
    FILE *arq;  
    arq = fopen("nome do arquivo", "modo");  
    ...  
    fclose(arq);  
    return 0;  
}
```

ABRIR E FECHAR UM ARQUIVO

- O **nome do arquivo** é um texto (ou variável tipo `char *`) que contém o nome do arquivo com o qual o programa deseja operar.

```
int main () {  
    FILE *arq;  
    arq = fopen("nome do arquivo", "modo");  
    ...  
    fclose(arq);  
    return 0;  
}
```

O nome pode incluir o caminho completo do arquivo.
Exemplo: (No Linux) `"/usr/arquivo"` é um nome válido.

ABRIR E FECHAR UM ARQUIVO

- O **modo** é um texto (ou variável tipo `char *`) que informa quais operações serão realizadas sobre o arquivo.

```
int main () {  
    FILE *arq;  
    arq = fopen("nome do arquivo", "modo");  
    ...  
    fclose(arq);  
    return 0;  
}
```

ABRIR E FECHAR UM ARQUIVO

- O modo de abrir um arquivo deve ser exatamente uma das seguintes opções:

Modo	Operações	Acesso	Ponto do arquivo	Observações
r (<i>read</i>)	Leitura	Sequencial	Início	Se o arquivo não existir ou não puder ser encontrado, então fopen falha.
w (<i>write</i>)	Escrita	Sequencial	Início	Cria o arquivo, se necessário. Se o arquivo já existir, seu conteúdo anterior será apagado .
a (<i>append</i>)	Escrita	Sequencial	Final	Cria o arquivo se necessário. Se o arquivo já existir, mantém o conteúdo anterior .

ABRIR E FECHAR UM ARQUIVO

- O modo de abrir um arquivo deve ser exatamente uma das seguintes opções:

Modo	Operações	Acesso	Ponto do arquivo	Observações
r+	Leitura e escrita	Aleatório	Início	Se o arquivo não existir ou não puder ser encontrado, então fopen falha.
w+	Leitura e escrita	Aleatório	Início	Cria o arquivo, se necessário. Se o arquivo já existir, seu conteúdo anterior será apagado .
a+	Leitura e escrita	Leitura: aleatório. Escrita: sequencial.	Leitura: início. Escrita: Final.	Cria o arquivo se necessário. Se o arquivo já existir, mantém o conteúdo anterior .

ABRIR E FECHAR UM ARQUIVO

- A função **fclose** fecha o arquivo. O único argumento de `fclose` é a variável com a referência do fluxo que desejamos fechar.

```
int main () {  
    FILE *arq;  
    arq = fopen("nome do arquivo", "modo");  
    ...  
    fclose(arq);  
    return 0;  
}
```

ABRIR E FECHAR UM ARQUIVO

```
#include<stdio.h>
void main(){

    FILE *arq; //declaração

    //Abre o arquivo em modo leitura
    arq = fopen("exemplo.txt", "r");

    if (arq == NULL) // Testa a abertura
        perror("Erro ao abrir o arquivo.\n");
    else
        printf("Arquivo aberto para leitura.\n");

    fclose(arq); // Libera o apontador
}
```

ABRIR E FECHAR UM ARQUIVO

```
#include<stdio.h>
void main(){

    FILE *arq; //declaração

    //Abre o arquivo em modo leitura
    arq = fopen("exemplo.txt", "r");

    if (arq == NULL) // Testa a abertura
        perror("Erro ao abrir o arquivo.\n");
    else
```

A função **perror()** obtém e exibe uma mensagem explicativa referente ao erro.

```
}
```

LER DO ARQUIVO

```
fscanf(arquivo, "formato", &variável);
```

Arquivo: A referência para o arquivo que será lido.

Formato: %d, %f, %c, %s, etc, para informar o que desejamos ler.

Variável: Uma ou mais variáveis (separadas por vírgula), que armazenarão os valores lidos.

LER DO ARQUIVO

```
fscanf(arquivo, "formato", &variável);
```

Arquivo: A referência para o arquivo que será lido.

Formato: %d, %f, %c, %s, etc, para informar o que desejamos ler.

Variável: Uma ou mais variáveis (separadas por vírgula), que armazenarão os valores lidos.

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

A função **fgetc** (*get character*) retorna o próximo caractere do arquivo e avança a leitura em uma posição.

LER DO ARQUIVO

```
fscanf(arquivo, "formato", &variável);
```

Arquivo: A referência para o arquivo que será lido.

Formato: %d, %f, %c, %s, etc, para informar o que desejamos ler.

Variável: Uma ou mais variáveis (separadas por vírgula), que armazenarão os valores lidos.

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

A função **fgetc** (*get character*) retorna o próximo caractere do arquivo e avança a leitura em uma posição.

```
fgets(linha, 100, arquivo);
```

Lê uma linha de texto (ou até atingir o limite de bytes lidos) e guarda o texto lido em **linha**. Retorna dest em caso de sucesso, NULL em caso de erro.

LER DO ARQUIVO

```
#include<stdio.h>
void main(){

    FILE *arq;
    arq = fopen("exemplo.txt", "r");

    if (arq == NULL)
        perror("Erro ao abrir o arquivo.\n");

    while (fscanf(f, "%d", &num) != EOF)
        printf("%d ", num);

    fclose(arq);
}
```

VERIFICANDO FIM DO ARQUIVO

A função **fEOF** retorna verdadeiro se a última operação (fscanf, fgetc ou fgets) falhou por não haver mais dados para serem lidos:

```
while (!feof(arquivo)) {  
    ...  
    Operação de leitura  
    ...  
}
```

A função fscanf retorna quantas variáveis foram lidas. Este resultado pode ser utilizado para decidir se a leitura deve ou não ser interrompida:

```
while(fscanf(arquivo, "%d %d %d", &a, &b, &c) != 0) {  
    ...  
}
```

ESCREVER NO ARQUIVO

```
fprintf(arquivo, "formato", variável);
```

Arquivo: A referência para o arquivo no qual se vai escrever.

Formato: %d, %f, %c, %s, etc, para informar o que desejamos escrever.

Variável: Uma ou mais variáveis (separadas por vírgula), cujos valores serão formatados de acordo com os respectivos indicadores.

ESCREVER NO ARQUIVO

```
fprintf(arquivo, "formato", variável);
```

Arquivo: A referência para o arquivo no qual se vai escrever.

Formato: %d, %f, %c, %s, etc, para informar o que desejamos escrever.

Variável: Uma ou mais variáveis (separadas por vírgula), cujos valores serão formatados de acordo com os respectivos indicadores.

```
fputc(c, arquivo);
```

A função **fputc** (*put character*) escreve no arquivo referenciado pela variável "**arquivo**" o caractere armazenado na variável **c**.

ESCREVER NO ARQUIVO

```
fprintf(arquivo, "formato", variável);
```

Arquivo: A referência para o arquivo no qual se vai escrever.

Formato: %d, %f, %c, %s, etc, para informar o que desejamos escrever.

Variável: Uma ou mais variáveis (separadas por vírgula), cujos valores serão formatados de acordo com os respectivos indicadores.

```
fputc(c, arquivo);
```

A função **fputc** (*put character*) escreve no arquivo referenciado pela variável "**arquivo**" o caractere armazenado na variável **c**.

```
fputs(texto, arquivo);
```

```
fputs("Conteudo de outro texto", arquivo);
```

A função **fputs** escreve uma linha inteira no arquivo. Retorna EOF em caso de erro.

ESCREVER NO ARQUIVO

```
FILE  *fr, *fw;
int  num;

fr = fopen ("teste.txt", "r");
fw = fopen ("saida.txt", "w");

if (fr == NULL){
    perror("teste.txt");
}

if (fw == NULL){
    perror("saida.txt");
}

while (fscanf(fr, "%d", &num) != EOF)
    fprintf(fw, "%d ", num);

fclose(fr);
fclose(fw);
```

AVANÇAR E RETROCEDER A POSIÇÃO DE LEITURA OU ESCRITA

- Reposicionar o “indicador” no início do arquivo:

```
void rewind(FILE *f)
```

- Consultar a posição atual da leitura ou da escrita (O valor retornado pela função `ftell` é sempre do tipo `long int`):

```
posicao = ftell(arquivo);
```


AVANÇAR E RETROCEDER A POSIÇÃO DE LEITURA OU ESCRITA

- Os arquivos de acesso aleatório podem mudar a posição de leitura e escrita com a função `fseek`:

```
fseek(arquivo, deslocamento, referencia);
```

Deslocamento: quantos caracteres desejamos avançar (se positivo) ou retroceder (se negativo) em relação à posição referencial.

Posição referencial: de onde o deslocamento deve ser realizado:

- **SEEK_CUR:** deslocamento em relação à posição de leitura e escrita atual.
- **SEEK_END:** deslocamento em relação ao final do arquivo.
- **SEEK_SET:** 0 deslocamento é em relação ao início do arquivo.

OUTRAS FUNÇÕES

Remove o arquivo com nome informado. Retorna 0 em caso de sucesso:

```
int remove(const char *filename)
```

Renomeia arquivo. Retorna 0 em caso de sucesso:

```
int rename(const char *old, const char *new)
```

Ex: `rename("arq1.txt", "arq2.txt");`

PARÂMETROS DA FUNÇÃO MAIN()

A função **main()** pode ter parâmetros formais. Mas o programador não pode escolher quais serão eles. Os parâmetros que função **main()** aceita são:

```
int main (int argc, char *argv[]);
```

argc contém o número de parametros passados.

***argv[]** contém todos os parametros passados (inclusive o nome do programa).

Exemplo de passagem de parâmetros:

```
./meuprograma 192.168.0.1 80
```

argc = 3

arg[0] = ./meuprograma

arg[1] = 192.168.0.1

arg[2] = 80

PARÂMETROS DA FUNÇÃO MAIN()

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

void main(int argc, char *argv[]){

    if(argc != 3){
        printf("Uso: <./nome_do_programa> <ip> <porta>");
        exit(-1);
    }

    printf("%s\n", argv[0]);
    printf("%s\n", argv[1]);
    printf("%s\n", argv[2]);
}
```

```
~$ ./meuprograma 192.168.0.1 80
./meuprograma
192.168.0.1
80
```