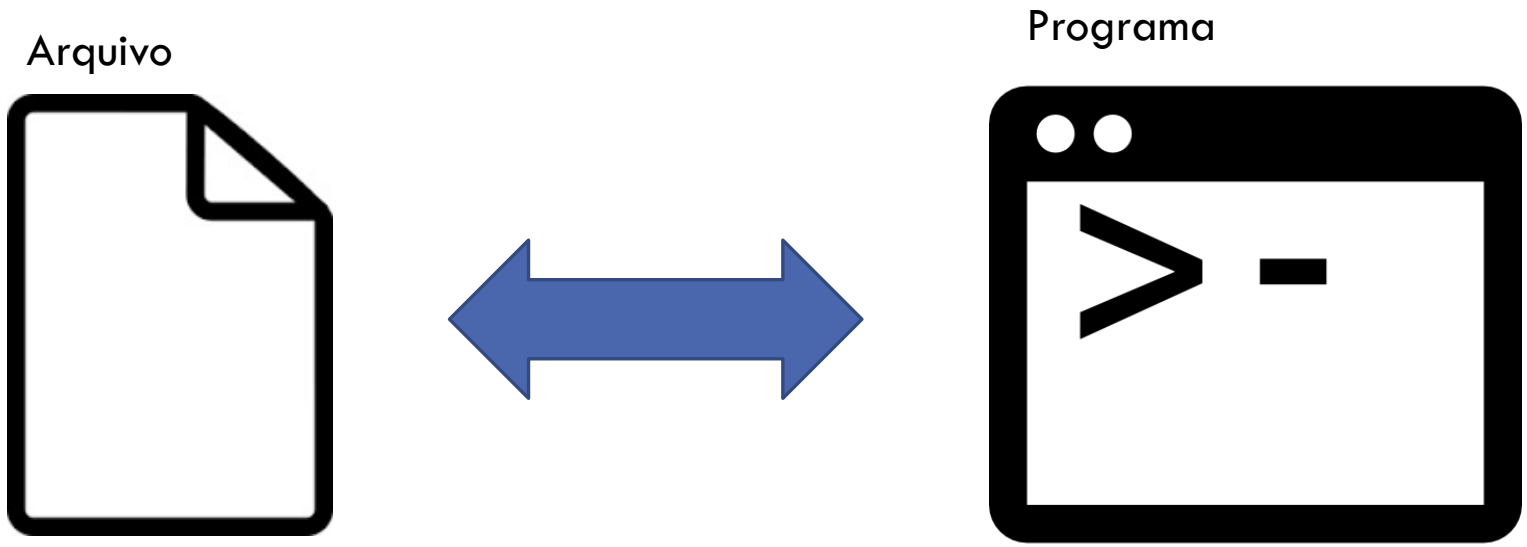
ARQUIVOS

Vanessa Braganholo

Estruturas de Dados e Seus Algoritmos

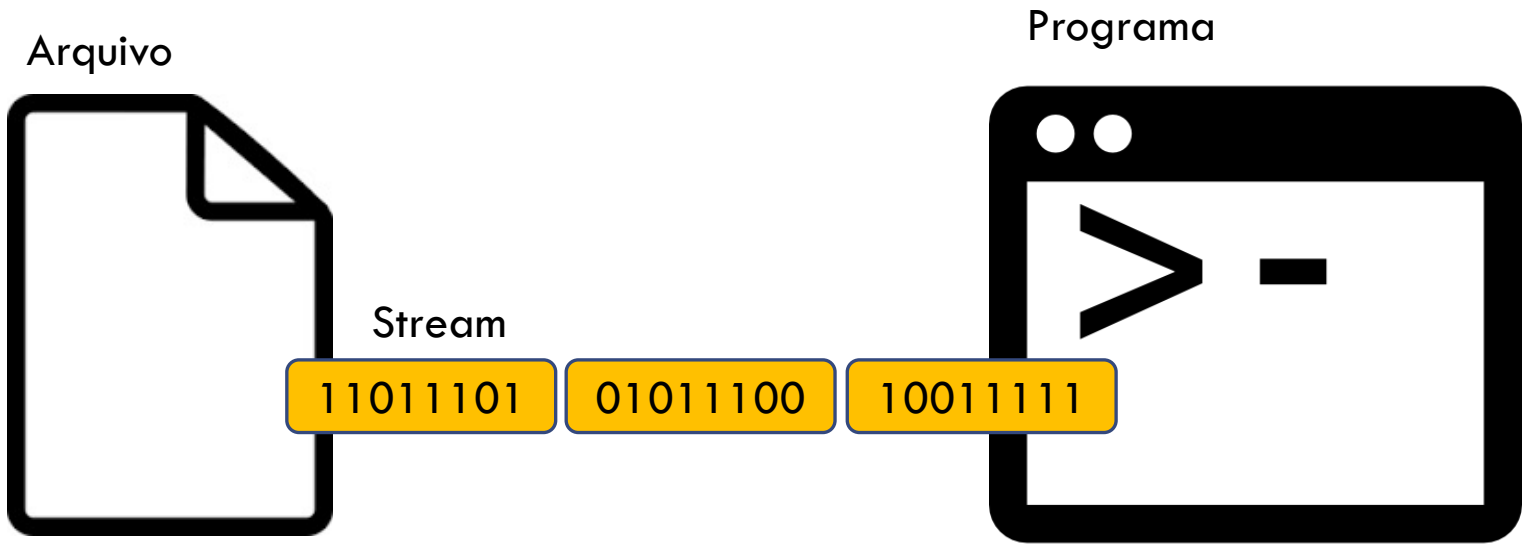
ARQUIVO

Arquivo é um conjunto de dados, dispostos de forma sequencial

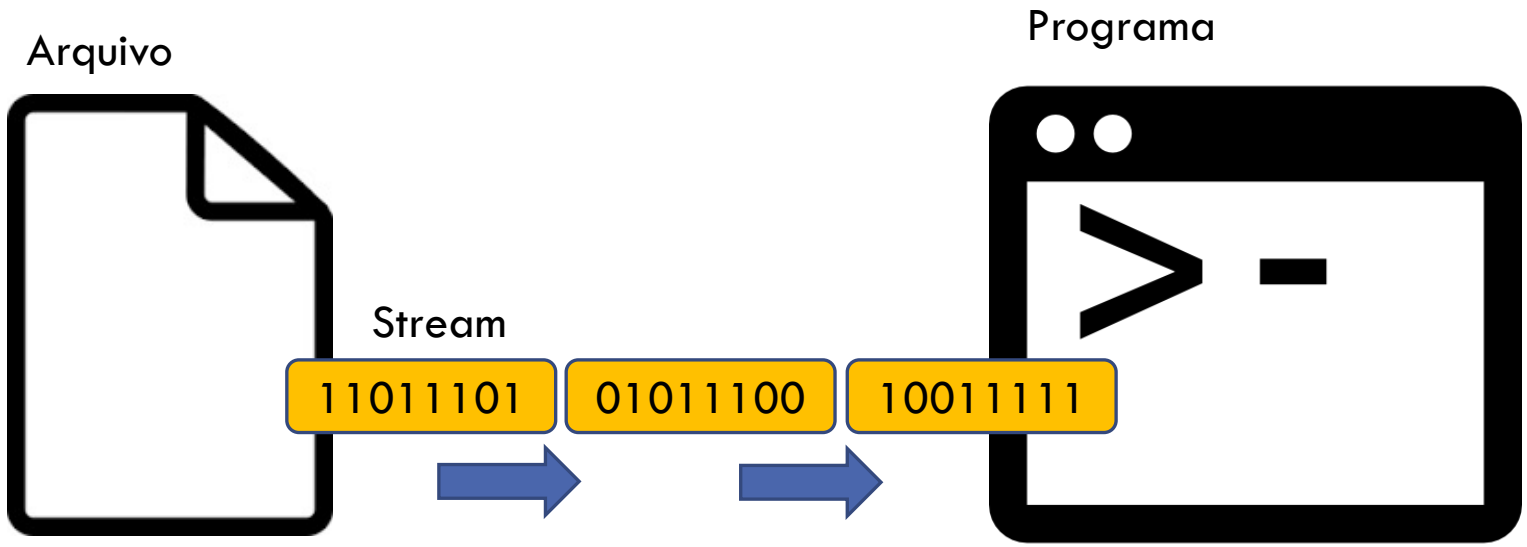


STREAM

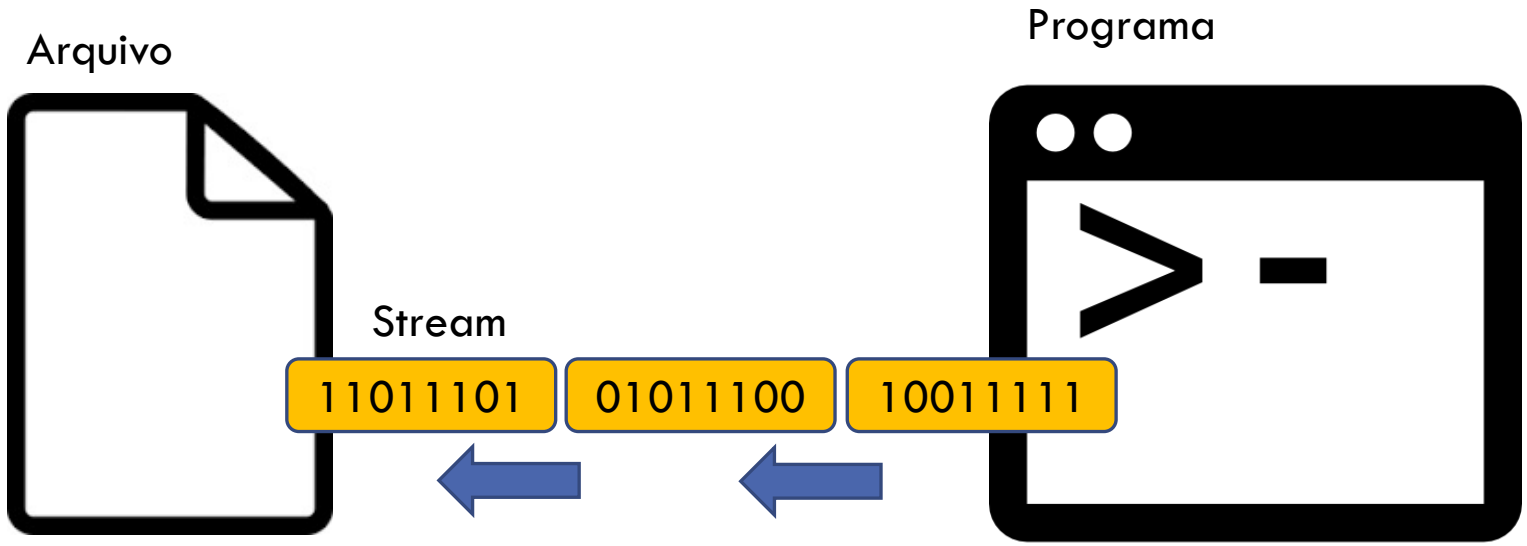
Leitura e escrita em um arquivo é feita por meio de um **stream**



LEITURA DE ARQUIVOS

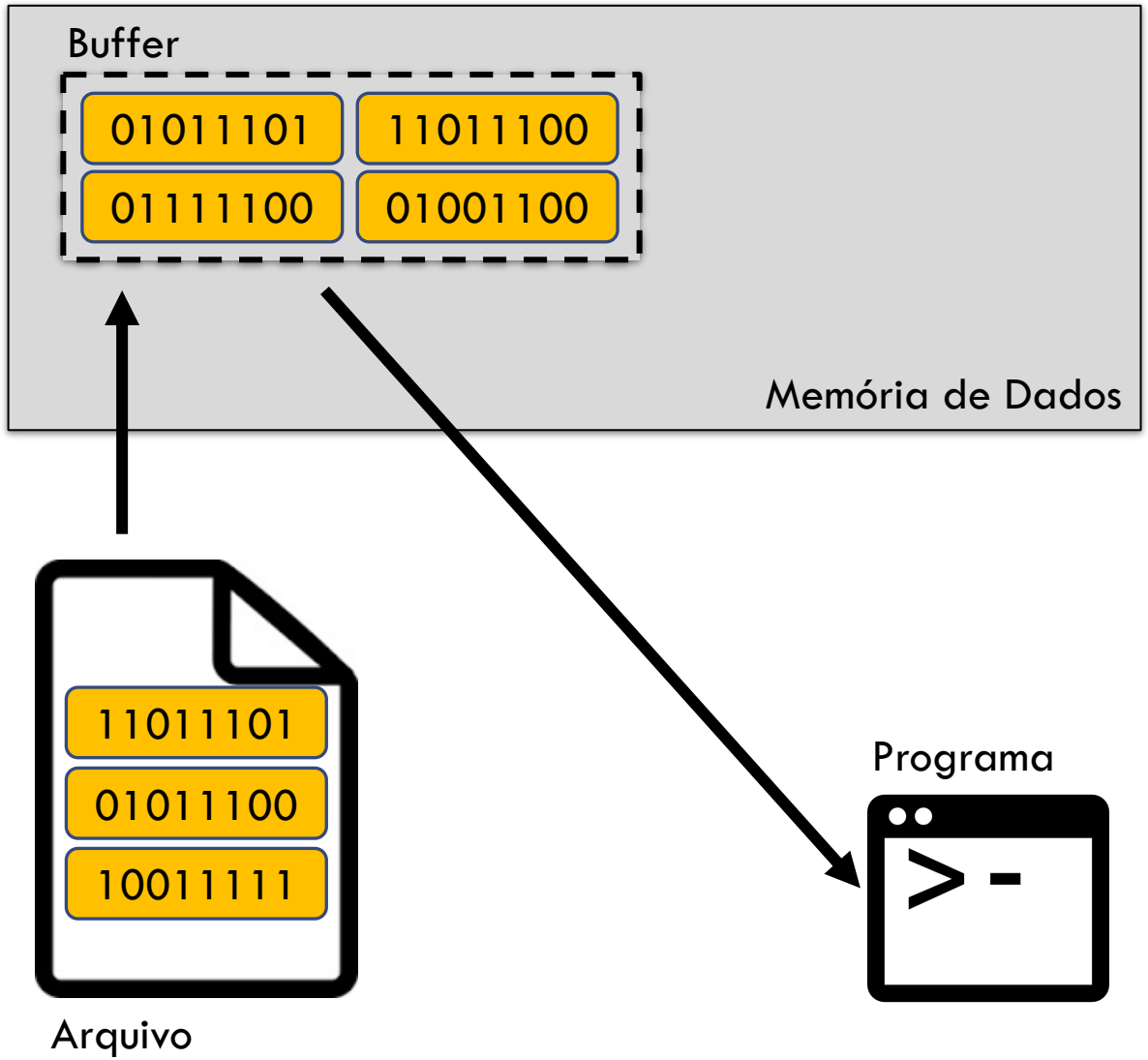


ESCRITA DE ARQUIVOS



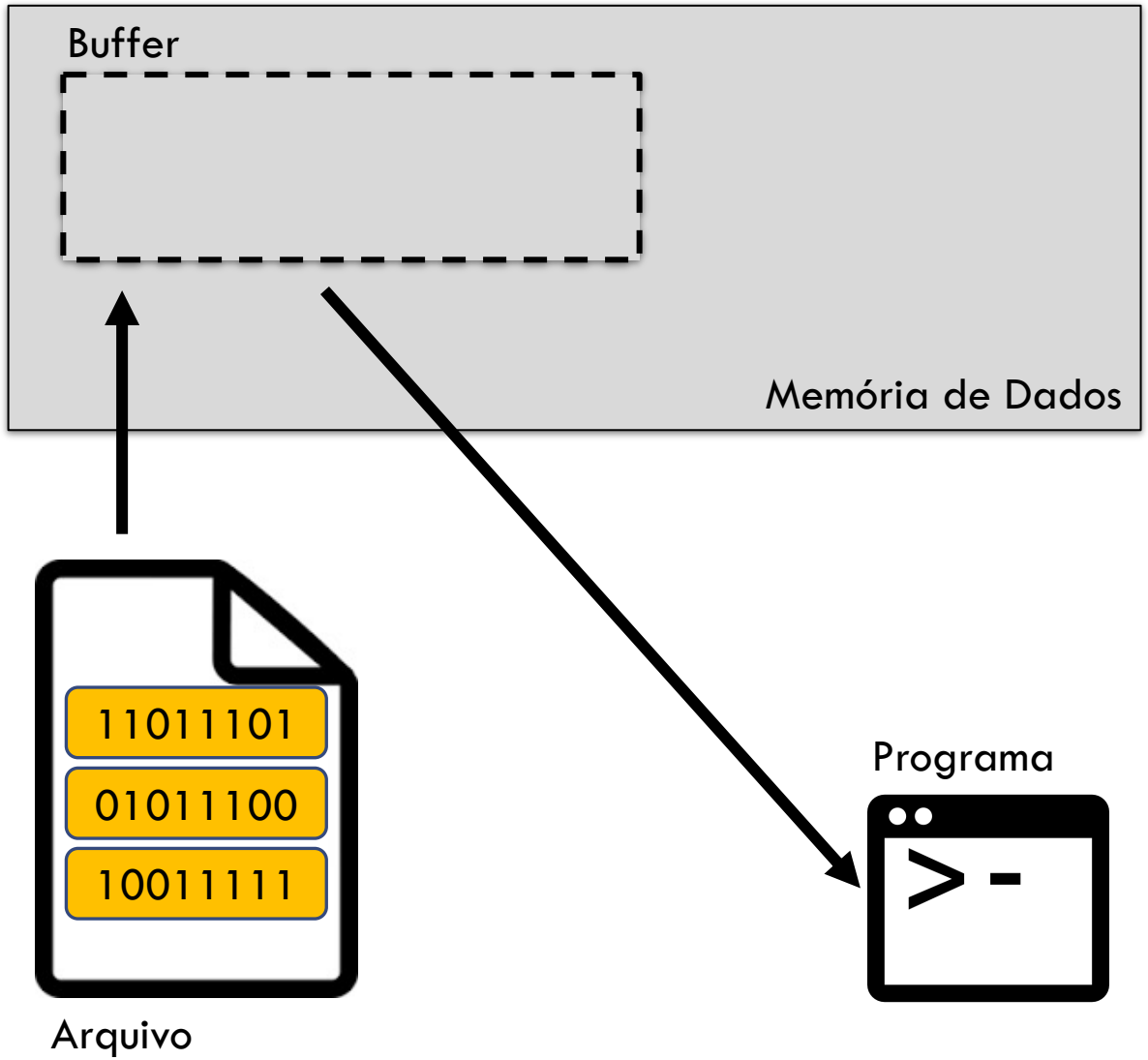
BUFFER

Um buffer pode ser usado para acelerar a **leitura** e **escrita** de arquivos



BUFFER PARA LEITURA

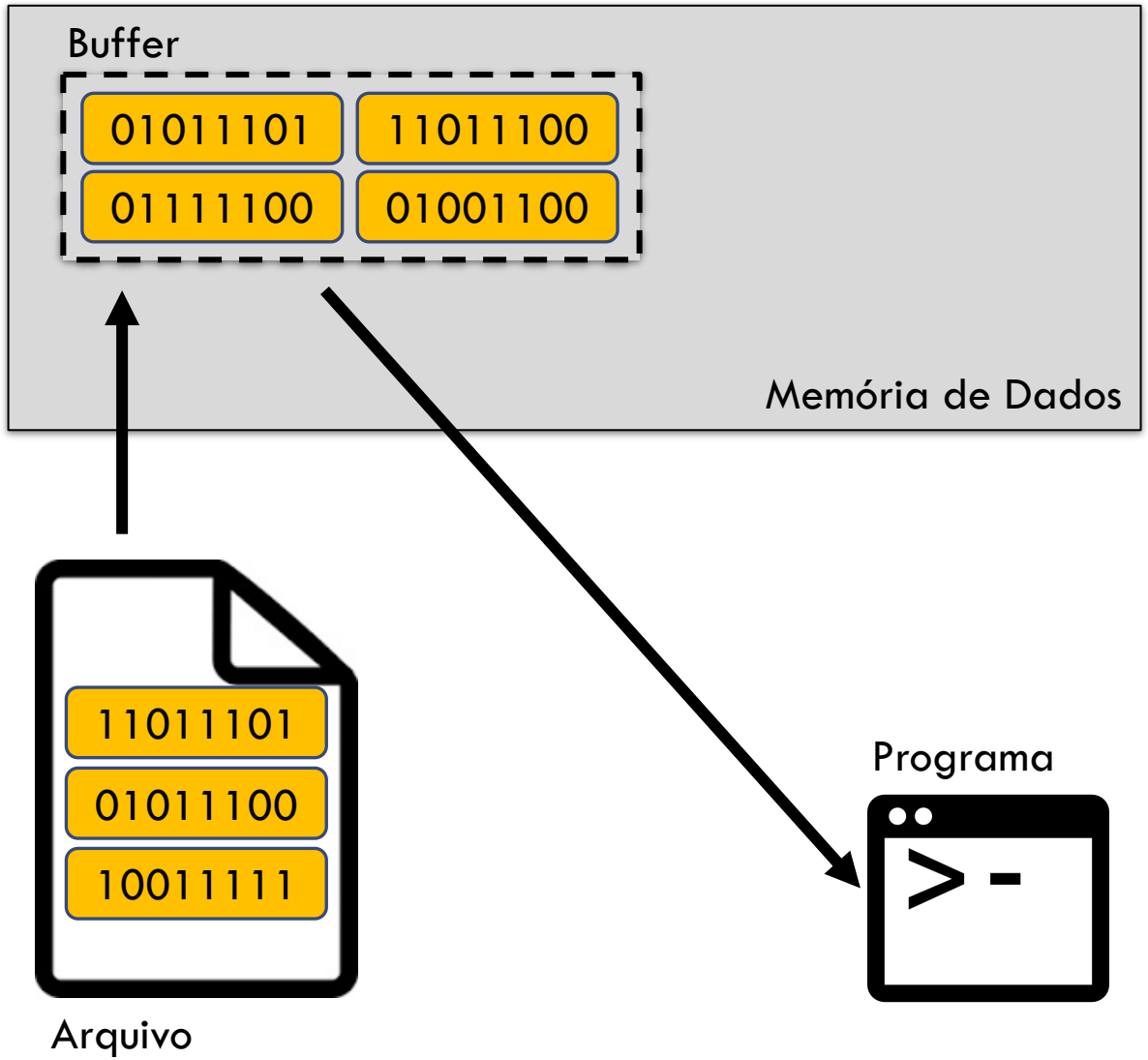
Situação inicial: buffer vazio



BUFFER PARA LEITURA

Programa faz operação de leitura

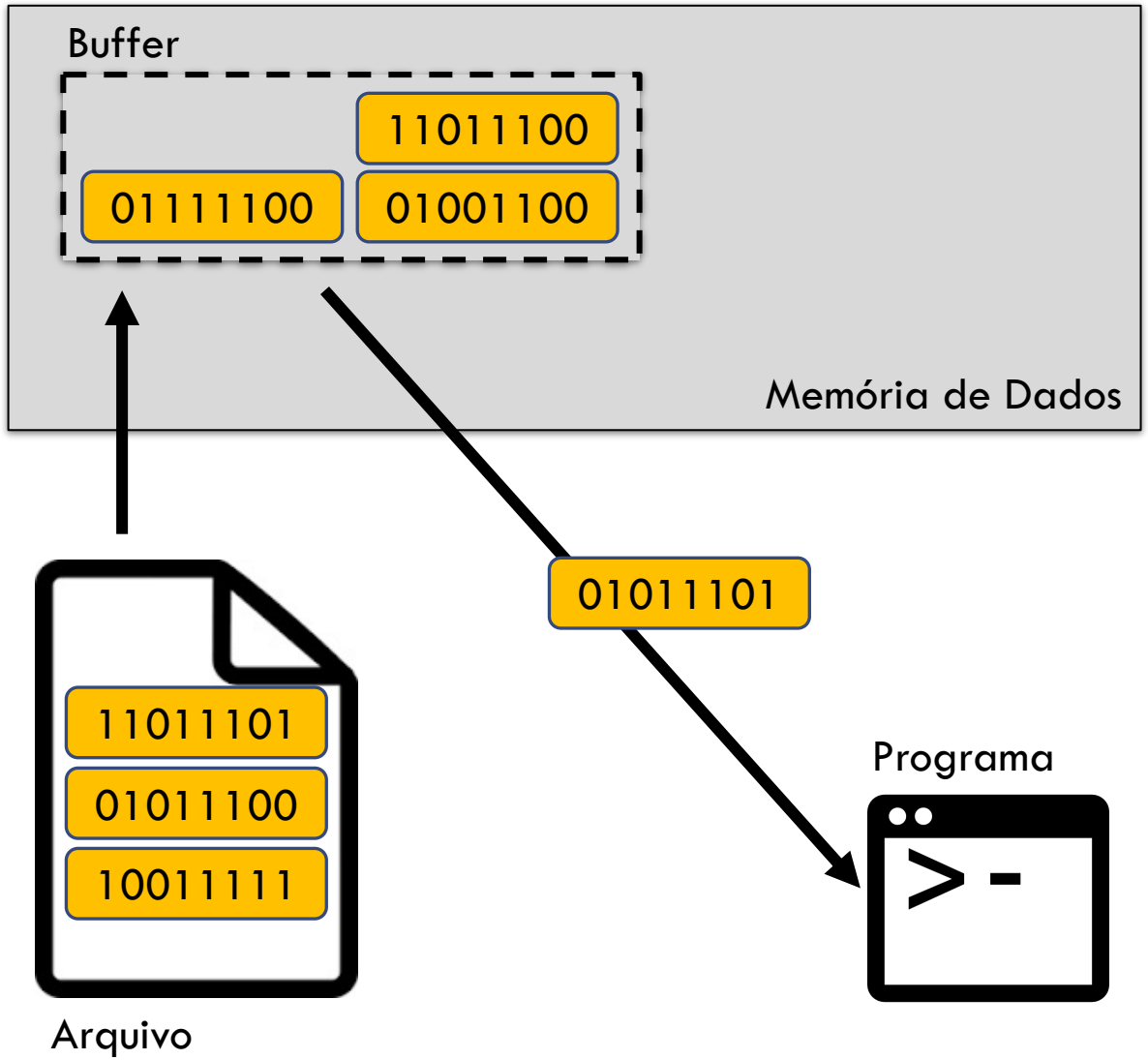
- Buffer está vazio, então dado é lido do arquivo para o buffer, até que ele fique cheio (ou que o arquivo acabe)



BUFFER PARA LEITURA

Programa faz operação de leitura

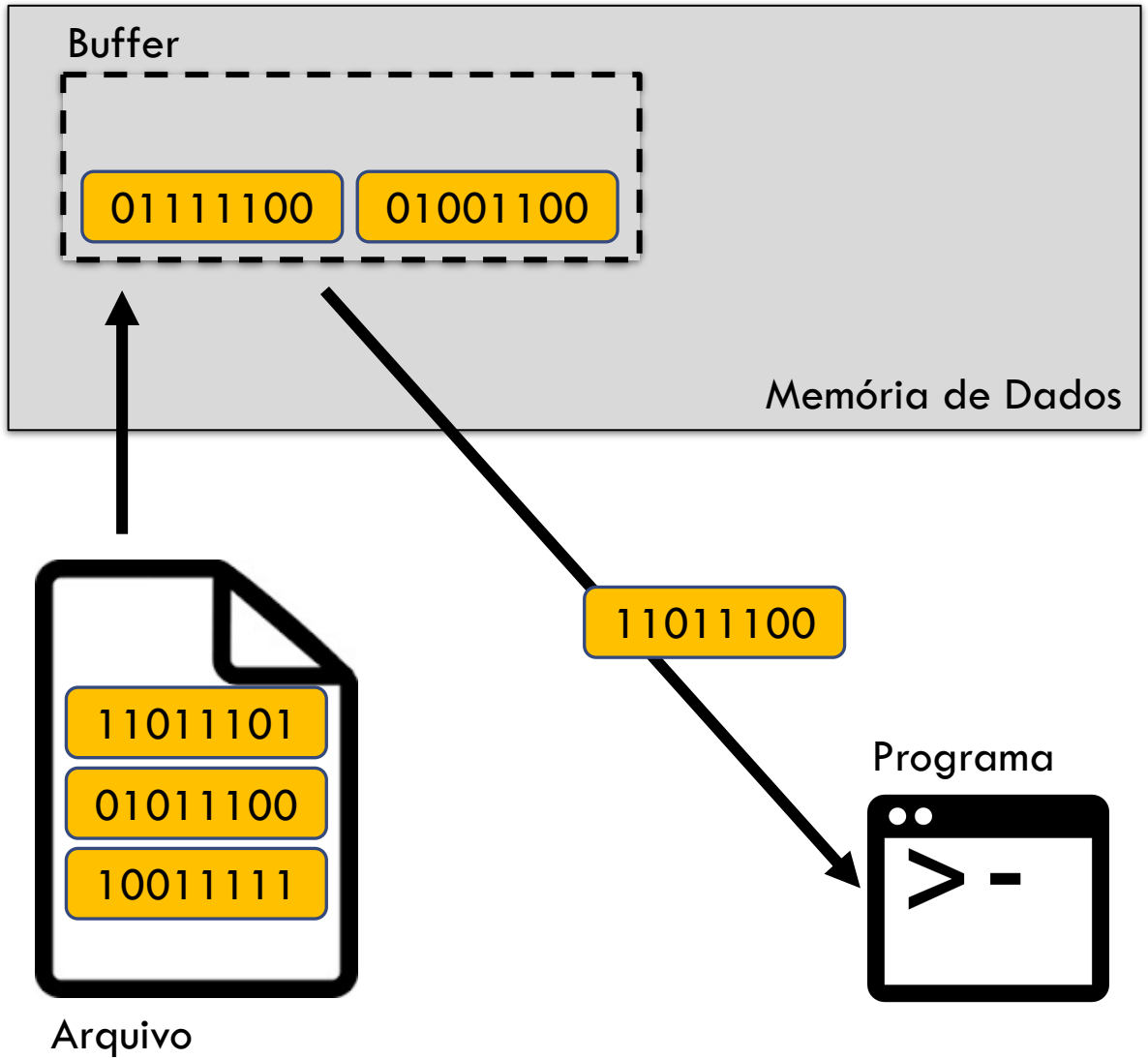
- Buffer está vazio, então dado é lido do arquivo para o buffer, até que ele fique cheio (ou que o arquivo acabe)
- Dado requisitado na leitura é enviado ao programa a partir do buffer (usando o stream que está conectado ao arquivo)



BUFFER PARA LEITURA

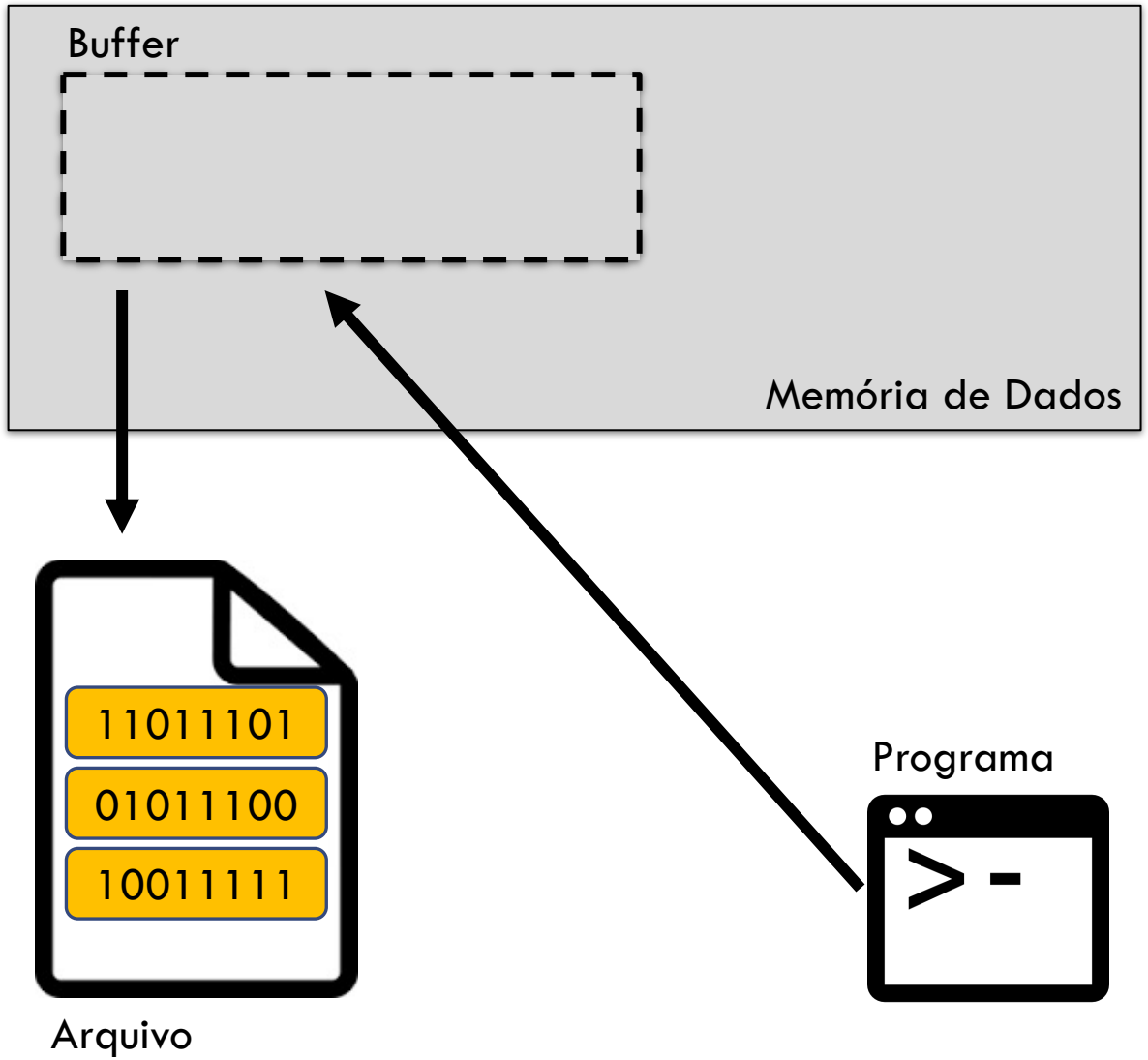
Programa faz operação de leitura

- Buffer está vazio, então dado é lido do arquivo para o buffer, até que ele fique cheio (ou que o arquivo acabe)
- Dado requisitado na leitura é enviado ao programa a partir do buffer (usando o stream que está conectado ao arquivo)



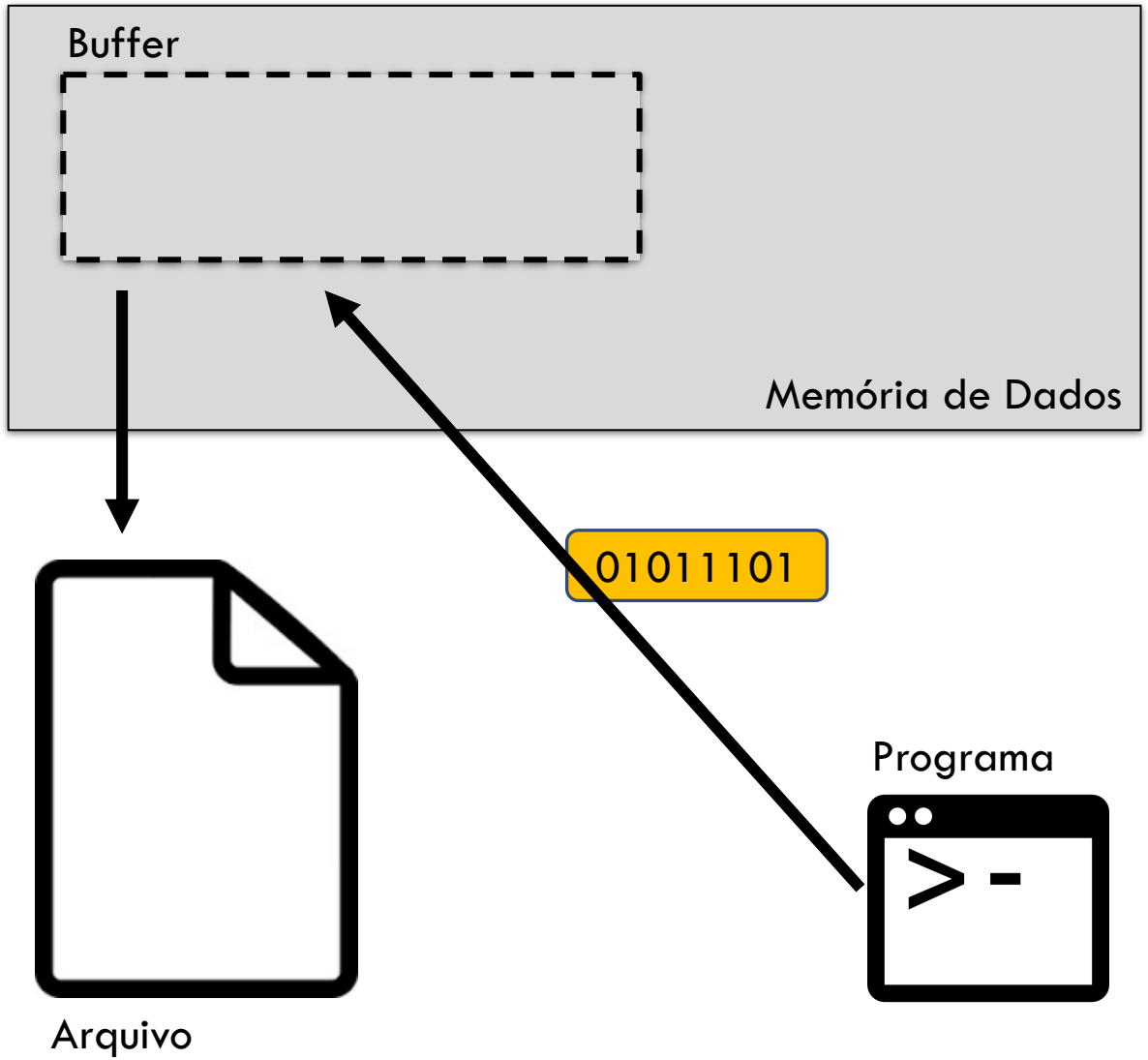
BUFFER PARA ESCRITA

Situação inicial: buffer vazio



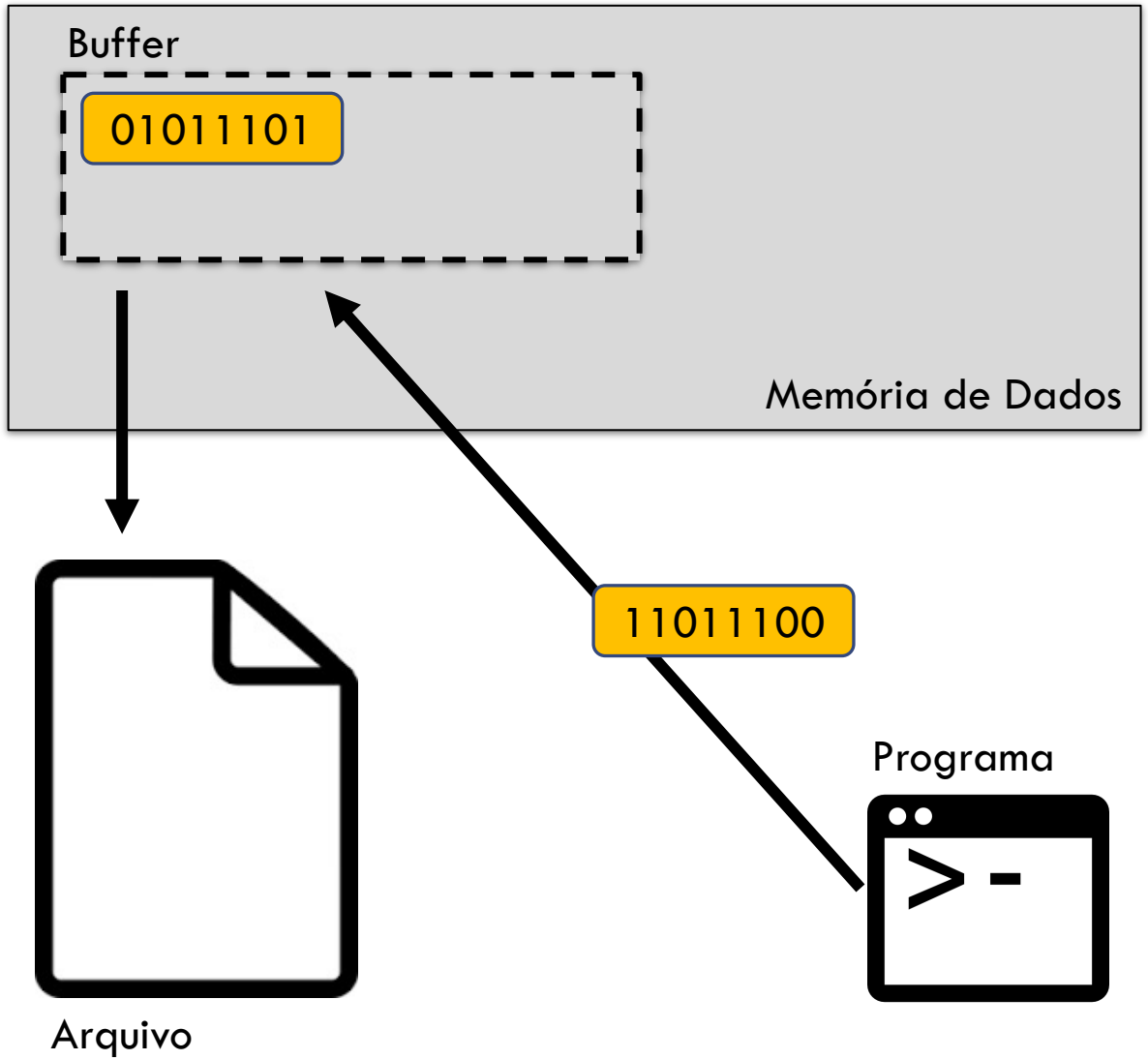
BUFFER PARA ESCRITA

Escrita vai sendo feita no buffer



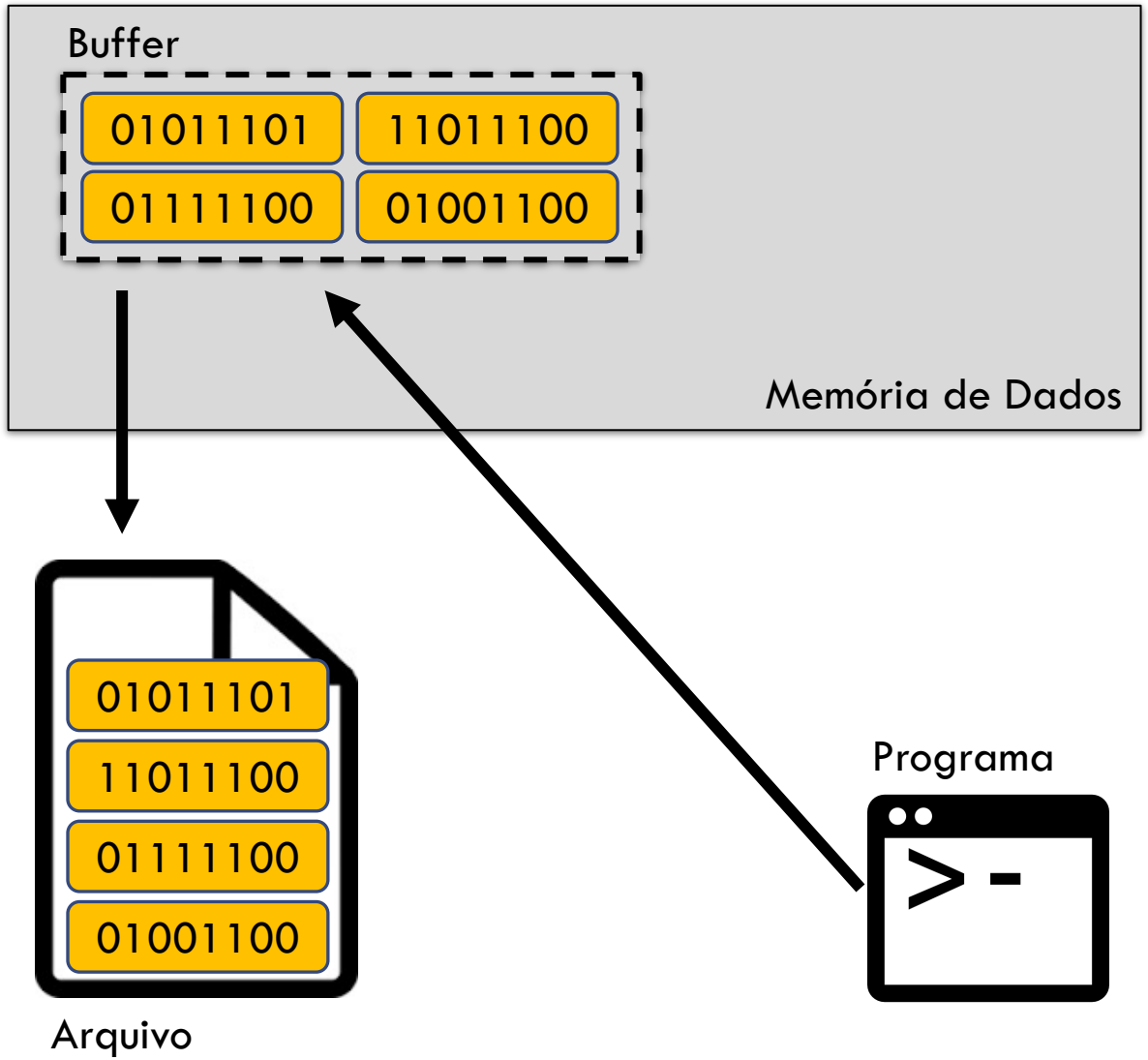
BUFFER PARA ESCRITA

Escrita vai sendo feita no buffer



BUFFER PARA ESCRITA

Quando o buffer enche ele é automaticamente descarregado no arquivo

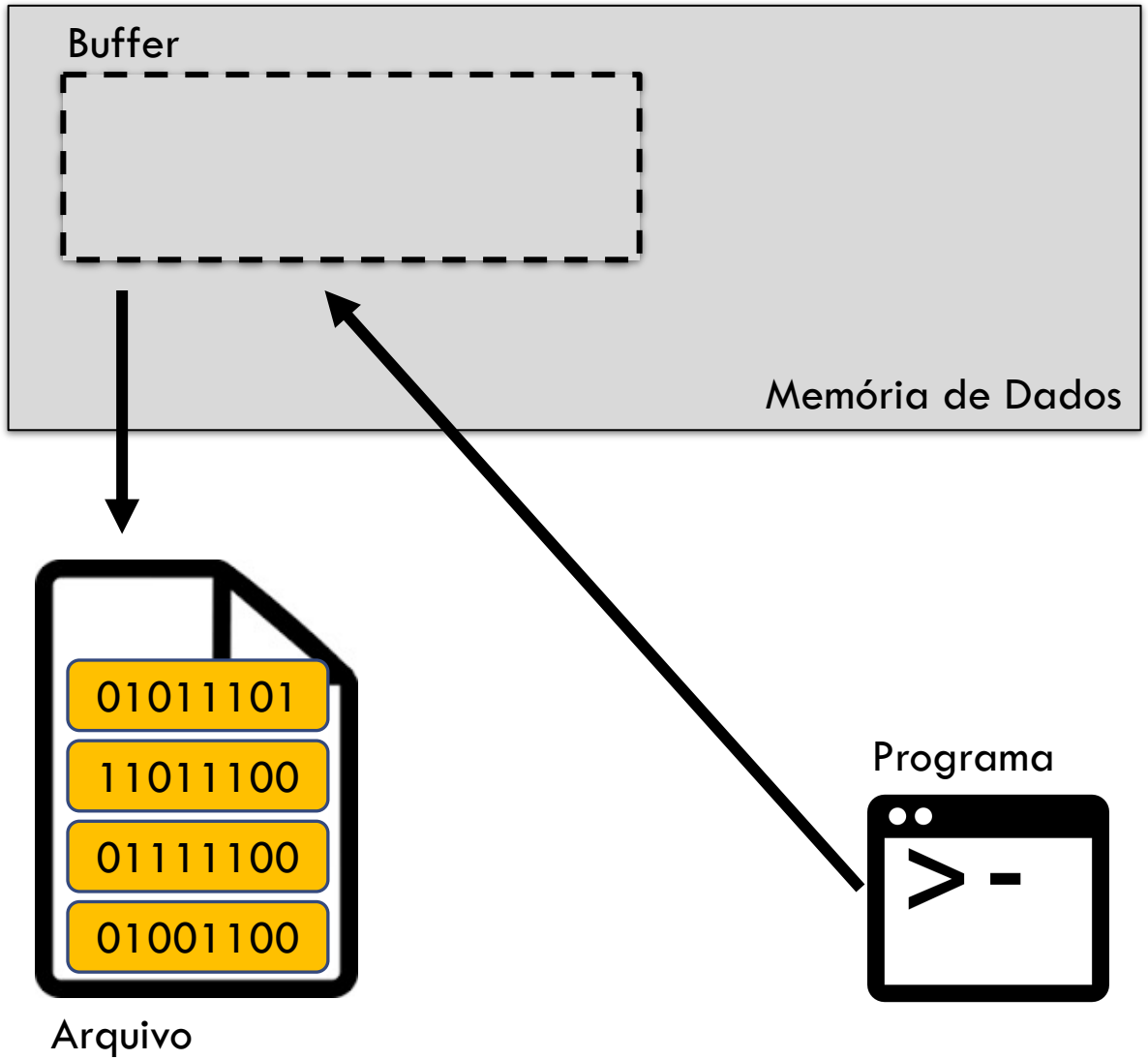


BUFFER PARA ESCRITA

Quando o buffer enche ele é automaticamente descarregado no arquivo

Depois é esvaziado para aguardar novas escritas...

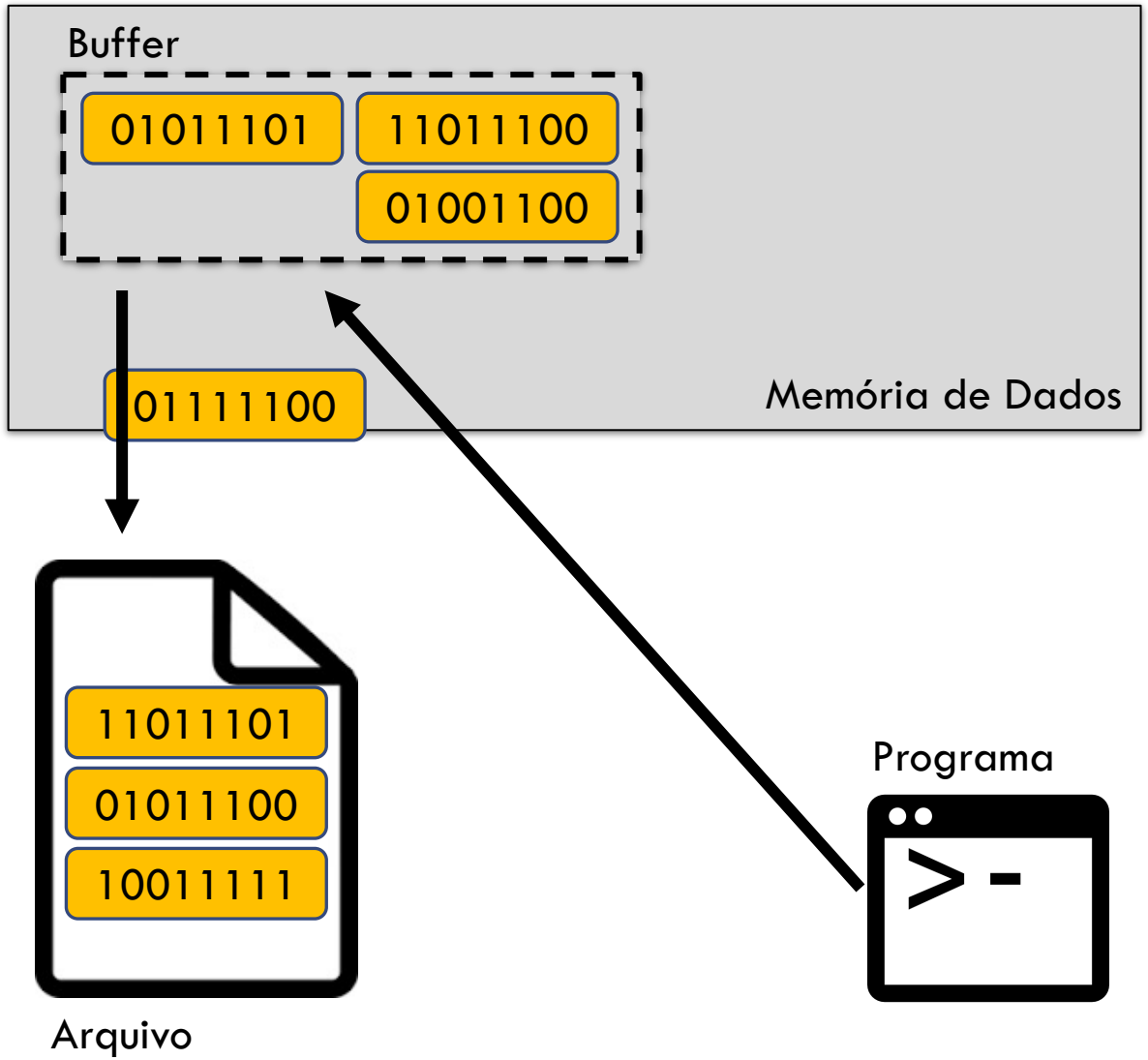
Essa operação se chama **flush**



FLUSH

Operação de flush descarrega **todo** o conteúdo do buffer

Flush pode ser **automático**, ou **forçado pelo programador** (mais detalhes mais adiante nessa aula)



CURSOR

Um cursor é associado ao arquivo de forma a indicar a próxima posição a ser lida ou gravada

O cursor é inicializado com 0 na abertura do arquivo

O cursor é incrementado a cada operação de leitura ou escrita no arquivo

TIPOS DE ARQUIVOS

Arquivo Texto

Arquivo Binário

ARQUIVO TEXTO

Conteúdo é apenas texto

ARQUIVO BINÁRIO

Conteúdo é binário

MANIPULAÇÃO DE ARQUIVOS EM C

Fonte:

Schildt, H. C Completo e Total. Ed. McGraw-Hill

ABERTURA DE ARQUIVO

Deve-se associar um **stream** a um arquivo e realizar uma operação de **abertura**

Após a abertura, informações podem ser trocadas entre o arquivo e o seu programa

A operação de abertura **inicializa o cursor**

FECHAMENTO DE ARQUIVO

A operação de fechamento de arquivo **desassocia o arquivo do stream**

Libera a memória (equivalente ao **free** p/ memória alocada dinamicamente)

Se um arquivo aberto para escrita for fechado, o conteúdo de seu buffer associado é escrito no arquivo para evitar perda de conteúdo

FILE

Em C, cada stream associado a um arquivo tem uma estrutura de controle de arquivo do tipo **FILE**

Essa estrutura é definida no cabeçalho **stdio.h**, que deve ser incluído em todos os programas que manipulam arquivos

FUNÇÕES

Nome	Função
fopen()	Abre um arquivo
fclose()	Fecha um arquivo
feof()	Devolve **verdadeiro** se o fim do arquivo for atingido
ferror()	Devolve **verdadeiro** se ocorreu erro
remove()	Apaga um arquivo
fflush()	Descarrega o buffer no arquivo

EXEMPLO

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

int main(int argc, char** argv) {
    FILE *arq; //declara ponteiro para arquivo
    //abre arquivo
    arq = fopen("dados.txt", "r");
    if (arq != NULL){// checa se não deu erro na abertura do arquivo
        //processa arquivo
        ...
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

EXEMPLO

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

int main(int argc, char** argv) {
    FILE *arq; //declara ponteiro para arquivo
    //abre arquivo
    arq = fopen("dados.txt", "r");
    if (arq != NULL){// checa se não deu erro na abertura do arquivo
        //processa arquivo
        ...
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

Nome do Arquivo

Modo de Abertura
(r = leitura de arquivo texto)

EXEMPLO

Se houver erro na abertura, **arq** ficará com valor **NULL**

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

int main(int argc, char** argv) {
    FILE *arq; //declara ponteiro para arquivo
    //abre arquivo
    arq = fopen("dados.txt", "r");
    if (arq != NULL){// checa se não deu erro na abertura do arquivo
        //processa arquivo
        ...
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

MANIPULAÇÃO DE ARQUIVOS TEXTO EM C

Fonte:

Schildt, H. C Completo e Total. Ed. McGraw-Hill

MODOS DE ABERTURA DE ARQUIVOS TEXTO

Modo	Significado
r	Abre um arquivo **texto** para leitura (se não existir, retorna NULL)
w	Abre um arquivo **texto** para escrita (se já existir, conteúdo é apagado, se não existir, será criado)
a	Abre (se já existir) ou cria um arquivo **texto** para escrita, preservando o conteúdo já existente
r+	Abre um arquivo **texto** para leitura e escrita (se arquivo não existe, retorna NULL)
w+	Cria e abre um arquivo **texto** para leitura e escrita (se arquivo já existe, conteúdo é apagado)
a+	Abre (se já existir) ou cria um arquivo **texto** para leitura e escrita – cursor é posicionado no final do arquivo

LEITURA E ESCRITA DE ARQUIVOS TEXTO

Nome	Função
fputc()	Escreve um caractere em um arquivo
fgetc()	Lê um caractere de um arquivo
fprintf()	É para um arquivo o que printf() é para o console
fscanf()	É para um arquivo o que scanf() é para o console

EXEMPLO – LEITURA CARACTERE A CARACTERE

```
void le_arquivo_caracteres() {
    FILE *arq; //declara ponteiro para arquivo
    //abre arquivo para leitura
    arq = fopen("../dados.txt", "r");

    if (arq != NULL){// checa se não deu erro na abertura do arquivo
        char c;
        while ((c = fgetc(arq)) != EOF) {//le char e testa se chegou ao fim
            printf("%c", c); //imprime caractere lido no monitor
        }
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

dados.txt

este é um arquivo texto
que tem várias linhas
1
2
3

EXEMPLO – LEITURA STRING A STRING

```
void le_arquivo_strings(){
    FILE *arq; //declara ponteiro para arquivo
    //abre arquivo para leitura
    arq = fopen("../dados.txt", "r");
    if (arq != NULL){// checa se não deu erro na abertura do arquivo
        char s[10];
        fscanf(arq, "%s", s);
        while (!feof(arq)) {//testa se chegou ao final do arquivo
            printf("%s\n", s);
            fscanf(arq, "%s", s);
        }
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

dados.txt

este é um arquivo texto
que tem várias linhas
1
2
3

EXEMPLO – LEITURA STRING A STRING

```
void le_arquivo_strings(){
    FILE *arq; //declara ponteiro para a
    //abre arquivo para leitura
    arq = fopen("../dados.txt", "r");
    if (arq != NULL){// checa se não deu
        char s[10];
        fscanf(arq, "%s", s);
        while (!feof(arq)) {//testa se chegou ao final do arquivo
            printf("%s\n", s);
            fscanf(arq, "%s", s);
        }
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

A leitura poderia ter sido feita direto dentro da condição do while como no exemplo anterior

```
char s[10];
while(fscanf(arq, "%s", s) != EOF) {
...
}
```

dados.txt

este é um arquivo texto
que tem várias linhas
1
2
3

EXEMPLO – LEITURA STRING A STRING COMO NÚMERO (CASO O ARQUIVO SÓ CONTENHA NÚMEROS)

```
void le_arquivo_strings(){
    FILE *arq; //declara ponteiro para arquivo
    //abre arquivo para leitura
    arq = fopen("../numeros.txt", "r");
    if (arq != NULL){// checa se não deu erro na abertura do arquivo
        int n;
        fscanf(arq, "%d", &n);
        while (!feof(arq)) {//testa se chegou ao final do arquivo
            printf("%d\n", n);
            fscanf(arq, "%d", &n);
        }
        fclose(arq); //fecha arquivo
    }
    else printf("Erro ao abrir arquivo\n");
}
```

numeros.txt

```
10 20
30
40
50 60 70
80
```

EXEMPLO – GRAVAÇÃO DE ARQUIVO TEXTO USANDO STRING

```
void grava_arquivo_strings(char* nomeArq) {
    FILE *arq; //declara ponteiro para arquivo

    //abre arquivo para gravação
    arq = fopen(nomeArq, "w");
    if (arq != NULL) {// checa se não deu erro na abertura do arquivo
        for (int i = 10; i < 100; i = i + 3) {
            fprintf(arq, "%d\n", i); //grava no arquivo
        }
        fclose(arq);
    }
    else printf("Erro ao abrir arquivo\n");
}
```

EXERCÍCIO

Dados dois arquivos texto contendo números dispostos de forma **ordenada**, gerar um arquivo equivalente ao **merge** dos dois arquivos, contendo todos os números presentes nos dois arquivos de entrada, mas **sem repetições**

```
void merge(char* nomeArq1, char* nomeArq2, char* nomeArqMerge)
```

ARQUIVOS BINÁRIOS

ENTIDADES

Aplicações precisam armazenar dados sobre as mais diversas **entidades,** que podem ser concretas ou abstratas

- Funcionário de uma empresa (concreto)
- Carros de uma locadora de veículos (concreto)
- Contas-corrente dos clientes de um banco (abstrato)
- Ligações telefônicas dos clientes de uma empresa de telefonia (abstrato)

ATRIBUTOS

Cada uma dessas entidades pode ser descrita por um conjunto de **atributos**

- Funcionário: nome, CPF, data-nascimento, salário
- Carro: marca, modelo, ano-fabricação, placa
- Conta-Corrente: agência, conta, saldo
- Ligações Telefônicas: data, origem, destino, duração

Os atributos também podem ser chamados de **campos**

REGISTROS

Indivíduos dessas entidades possuem um valor para cada um desses atributos (chamados de **pares atributo-valor**)

Um conjunto de pares atributo-valor que identifica um indivíduo de uma entidade é chamado de **registro**

EXEMPLOS DE REGISTROS

Funcionário:

<nome, João>, <CPF, 012345678-90>, <data-nascimento, 10/04/1980>, <salário, 3000>

Carro

<marca, Honda>, <modelo, Fit>, <ano-fabricação, 2010>, <placa, XYZ0123>

Conta-Corrente

<agencia, 0123>, <conta, 123456>, <saldo, 2000>

Ligação Telefônica

<data, 01/07/2010>, <origem, 21-2598-3311>, <destino, 21-2589-3322>, <duração, 10'36">

TABELA

Uma **tabela** é um conjunto ordenado de registros. Uma tabela pode ser armazenada em memória principal ou em memória secundária (disco)

Nesse segundo caso, também costuma ser chamada de **arquivo**

EXEMPLO: ARQUIVO DE FUNCIONÁRIOS

Nome	CPF	Data-Nascimento	Salário
João	012345678-90	10/04/1980	3000
Maria	234567890-12	25/07/1978	5000
Lúcia	345678901-23	27/04/1981	1500

IMPORTANTE: Todos os registros de uma mesma tabela possuem a mesma estrutura (mesmo conjunto de atributos/campos)

PROBLEMA: ENCONTRAR REGISTROS

Problema comum de diversas aplicações: encontrar um ou mais registros em uma tabela

- Encontrar o empregado de nome Maria
- Encontrar todos os empregados que ganham 3000
- Encontrar todos os empregados que nasceram em 27/04/1981

CONCEITO DE CHAVE

Dados usados para encontrar um registro: chave

Chave: subconjunto de atributos que identifica um determinado registro

CHAVE PRIMÁRIA E SECUNDÁRIA

Chave **primária**: subconjunto de atributos que identifica unicamente um determinado registro. Exemplo: CPF do funcionário ou RG do funcionário

- Na hipótese de uma chave primária ser formada por uma combinação de campos, essa combinação deve ser mínima (não deve conter campos supérfluos)
- Eventualmente, podemos encontrar mais de uma combinação mínima de campos que forma uma chave primária

Chave **secundária**: subconjunto de atributos que identificam um conjunto de registros de uma tabela. Exemplo: Nome do funcionário

MANIPULAÇÃO DE ARQUIVOS BINÁRIOS EM C

Fonte:

Schildt, H. C Completo e Total. Ed. McGraw-Hill

MODOS DE ABERTURA DE ARQUIVOS BINÁRIOS

Modo	Significado
rb	Abre um arquivo **binário** para leitura (se arquivo não existe, retorna NULL)
wb	Abre um arquivo **binário** para escrita (se arquivo já existe, conteúdo é apagado, se arquivo não existe, será criado)
ab	Abre um arquivo **binário** para escrita, preservando o conteúdo já existente (se arquivo não existe, será criado)
rb+	Abre um arquivo **binário** para leitura e escrita (se arquivo não existe, retorna NULL)
wb+	Cria um arquivo **binário** para leitura e escrita (se arquivo já existe, conteúdo é apagado)
ab+	Abre (se já existir) um arquivo **binário** para leitura e escrita, preservando o conteúdo já existente (escreve sempre no final do arquivo – append). Se não existir, cria arquivo.

FUNÇÕES PARA MANIPULAÇÃO DE ARQUIVOS BINÁRIOS

Nome	Função
fseek()	Posiciona o cursor em um byte específico
ftell()	Retorna a posição atual do cursor
rewind()	Posiciona o cursor no início do arquivo
fread()	Lê dado binário do arquivo
fwrite()	Escreve dado binário em arquivo
feof()	Devolve **verdadeiro** se o fim do arquivo for atingido
ferror()	Devolve **verdadeiro** se ocorreu erro
remove()	Apaga um arquivo
fflush()	Descarrega os dados do buffer p/ o arquivo

EXEMPLO

Gravar registros de funcionários num arquivo

STRUCT FUNCIONÁRIO

```
typedef struct Funcionario {
      int cod;
      char nome[50];
      char cpf[15];
      char data_nascimento[11];
      double salario;
} TFunc;
```

SALVA FUNCIONÁRIO

```
// Salva no arquivo out, na posição atual do cursor
void salva(TFunc *func, FILE *out) {
    fwrite(&func->cod, sizeof(int), 1, out);
    //func->nome ao invés de &func->nome,
    //pois string já é ponteiro
    fwrite(func->nome, sizeof(char),
           sizeof(func->nome), out);
    fwrite(func->cpf, sizeof(char),
           sizeof(func->cpf), out);
    fwrite(func->data_nascimento, sizeof(char),
           sizeof(func->data_nascimento), out);
    fwrite(&func->salario, sizeof(double), 1, out);
}
```

SALVA FUNCIONÁRIO

```
// Salva no arquivo out, na posição atual do cursor
void salva(TFunc *func, FILE *out) {
    fwrite(&func->cod, sizeof(int), 1, out);
    //func->nome ao invés de &func->nome,
    //pois string já é ponteiro
    fwrite(func->nome, sizeof(char),
           sizeof(func->nome), out);
    fwrite(func->cpf, sizeof(char),
           sizeof(func->cpf), out);
    fwrite(func->data_nascimento, sizeof(char),
           sizeof(func->data_nascimento), out);
    fwrite(&func->salario, sizeof(double), 1, out);
}
```

ORDEM

é importante – posteriormente, registro deverá ser lido nessa mesma ordem

SALVA FUNCIONÁRIO

```
// Salva no arquivo out, na posição atual do cursor
void salva(Funcionario *func, FILE *out) {
    fwrite(&func->cod, sizeof(int), 1, out);
    //func->nome ao invés de &func->nome,
    //pois string já é ponteiro
    fwrite(func->nome, sizeof(char),
                    ->nome), out);
    f               sizeof(char),
                    ->cpf), out);
    fwrite(func->data_nascimento, sizeof(char),
           sizeof(func->data_nascimento), out);
    fwrite(&func->salario, sizeof(doub
}
```

Dado a ser gravado

Arquivo destino

Tamanho em bytes do dado a ser gravado

Número de itens a serem gravados

LER UM REGISTRO DE FUNCIONÁRIO

```
// Le do arquivo in na posição atual do cursor
// Retorna um ponteiro para funcionário lido do arquivo
TFunc *le(FILE *in) {
    TFunc *func = (Funcionario *) malloc(sizeof(Funcionario));
    if (0 >= fread(&func->cod, sizeof(int), 1, in)) {
      free(func);
      return NULL;
    }
    fread(func->nome, sizeof(char), sizeof(func->nome), in);
    fread(func->cpf, sizeof(char), sizeof(func->cpf), in);
    fread(func->data_nascimento, sizeof(char),
                sizeof(func->data_nascimento), in);
    fread(&func->salario, sizeof(double), 1, in);
    return func;
}
```

LER UM REGISTRO DE FUNCIONÁRIO

Leitura na mesma ordem em que foi feita a gravação

```
// Le do arquivo in na posição atual do cursor
// Retorna um ponteiro para funcionário lido do arqu
TFunc *le(FILE *in) {
    TFunc *func = (Funcionario *) malloc(sizeof(Funcionario));
    if (0 >= fread(&func->cod, sizeof(int), 1, in)) {
      free(func);
      return NULL;
    }
    fread(func->nome, sizeof(char), sizeof(func->nome), in);
    fread(func->cpf, sizeof(char), sizeof(func->cpf), in);
    fread(func->data_nascimento, sizeof(char),
                sizeof(func->data_nascimento), in);
    fread(&func->salario, sizeof(double), 1, in);
    return func;
}
```

LER UM REGISTRO DE FUNCIONÁRIO

```
// Le do arquivo in na posição atual do cursor
// Retorna um ponteiro para funcionário lido do arquivo
TFunc *le(FILE *in) {
    TFunc *func = (Funcionario *) malloc(sizeof(Funcionario));
    if (0 >= fread(&func->cod, sizeof(int), 1, in)) {
      free(func);
      return NULL;
    }
    f                    eof(char), sizeof(func->nome
    f                    of(char), sizeof(func->cpf),
    f                    imento, sizeof(char),
                  sizeof(func->data_nascimento), in);
    fread(&func->salario, sizeof(double), 1, in);
    return func;
}
```

Variável para guardar dado a ser lido

Tamanho em bytes do dado a ser lido

Arquivo fonte

Número de itens a serem lidos

TIPOS DE ACESSO A ARQUIVOS

TIPOS DE ACESSO A ARQUIVOS

ARQUIVO DE ACESSO SEQUENCIAL

Arquivos texto só podem ser acessados via acesso **sequencial**

Leitura do arquivo é feita do início ao fim, string por string

Não é possível pular diretamente para um determinado ponto do arquivo

ARQUIVO DE ACESSO DIRETO

Arquivo em que o acesso a um registro pode ser feito diretamente, sem ter que ler todos os registros que vêm antes dele

Pode-se pular para um determinado registro usando **fseek**, bastando para isso saber o **endereço** dele

Possível usar apenas para **arquivos binários**

ENDEREÇO?

Como saber o endereço de um determinado registro?

Endereço é definido sempre como um **deslocamento** em relação a um ponto de partida (normalmente se usa o início do arquivo como referência)

EndereçoReg(i) = (i – 1) * tamanhoReg

onde **tamanhoReg** é o tamanho dos registros do arquivo, em bytes

EXEMPLO

CodCli	Nome	DataNascimento
10	Joao	02/12/1990
02	Maria	04/10/1976
15	Carlos	30/06/1979
04	Carolina	14/05/2000
01	Murilo	23/10/1988

Tamanho Registro:

- CodCli = 4 bytes
- Nome = 10 bytes
- DataNascimento = 12 bytes

- Total: 26 bytes por registro

EXEMPLO

	CodCli	Nome	DataNascimento
0	10	Joao	02/12/1990
26	02	Maria	04/10/1976
52	15	Carlos	30/06/1979
78	04	Carolina	14/05/2000
104	01	Murilo	23/10/1988

Endereço do Registro 3 = (3-1) * 26 = 52

Tamanho Registro:

- CodCli = 4 bytes
- Nome = 10 bytes
- DataNascimento = 12 bytes

- Total: 26 bytes por registro

PARA LER O REGISTRO 3

1. Abrir o arquivo
2. Calcular o endereço do registro 3
3. Avançar o cursor (**fseek**) para o endereço calculado
4. Ler o registro

Ao terminar de ler o registro, o cursor estará posicionado no registro 4

FSEEK

fseek(FILE *arq, long int **desloc**, int **flag**)

flag pode ser

- **SEEK_SET**: indica que deslocamento será feito a partir do início do arquivo
- **SEEK_CUR**: indica que deslocamento será feito a partir da posição atual do cursor
- **SEEK_END**: indica que deslocamento será feito a partir do final do arquivo
  - útil para descobrir tamanho do arquivo, combinado com o uso da função **ftell,** que retorna a posição atual do cursor, em bytes

EXEMPLO: DESCOBRIR O NÚMERO DE REGISTROS DE UM ARQUIVO

```
/*
 * tamanho do registro, retornado pela função
 * tamanho_registro() é dado em bytes
 */
int tamanho_arquivo(FILE *arq) {
    fseek(arq, 0, SEEK_END);
    int tam = trunc(ftell(arq) / tamanho_registro());
    return tam;
}
```

COLOCANDO TUDO EM AÇÃO...

Acompanhem o exemplo do tutorial de manipulação de arquivos em C, disponível no site da disciplina

REFERÊNCIA

Schildt, H. C Completo e Total. Ed. McGraw-Hill