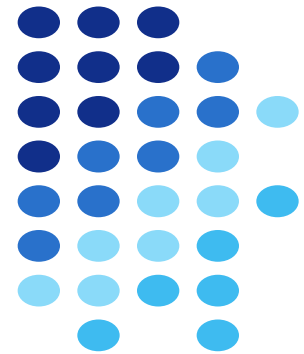


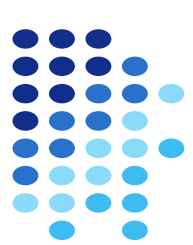
Universidade Federal de Sergipe
Departamento de Sistemas de Informação
SINF0007 – Estrutura de Dados II

Arquivos em C



2.1

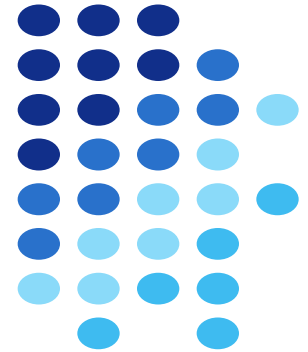
Prof. Dr. Raphael Pereira de Oliveira
raphael.oliveira@academico.ufs.br

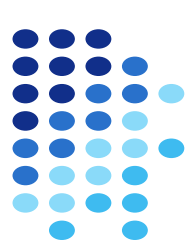


Arquivos

- Conceitos sobre Arquivos
- Manipulação de Arquivos em C
- Manipulação de Arquivos Texto em C
- Arquivos Binários
- Manipulação de Arquivos Binários em C
- Tipos de Acesso à Arquivos

Conceitos sobre Arquivos

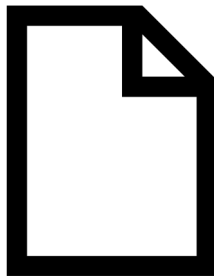




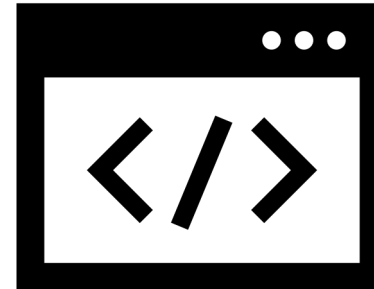
Arquivo

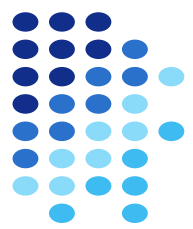
- Arquivo é um conjunto de dados dispostos de forma sequencial

Arquivo



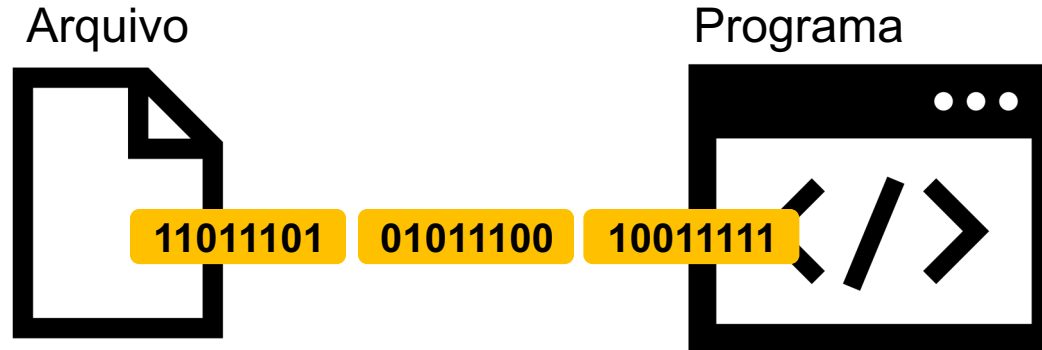
Programa

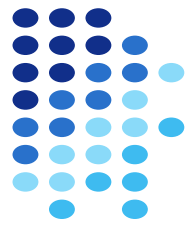




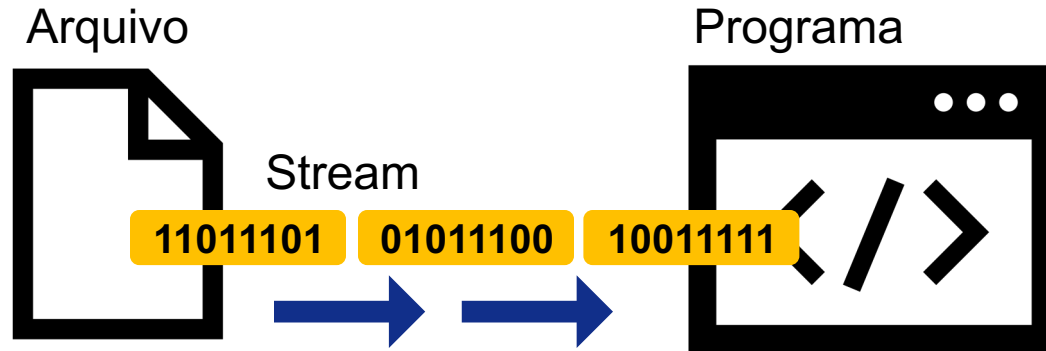
Stream

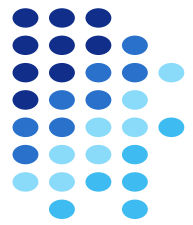
- Leitura e escrita em um arquivo é feita por meio de um **stream**



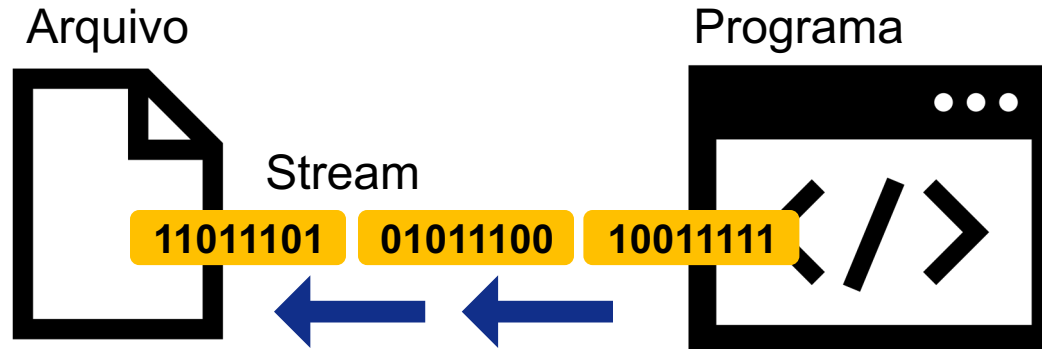


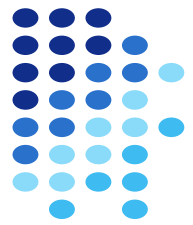
Leitura de Arquivos





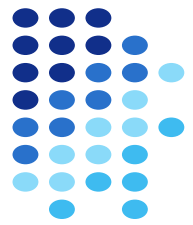
Escrita de Arquivos





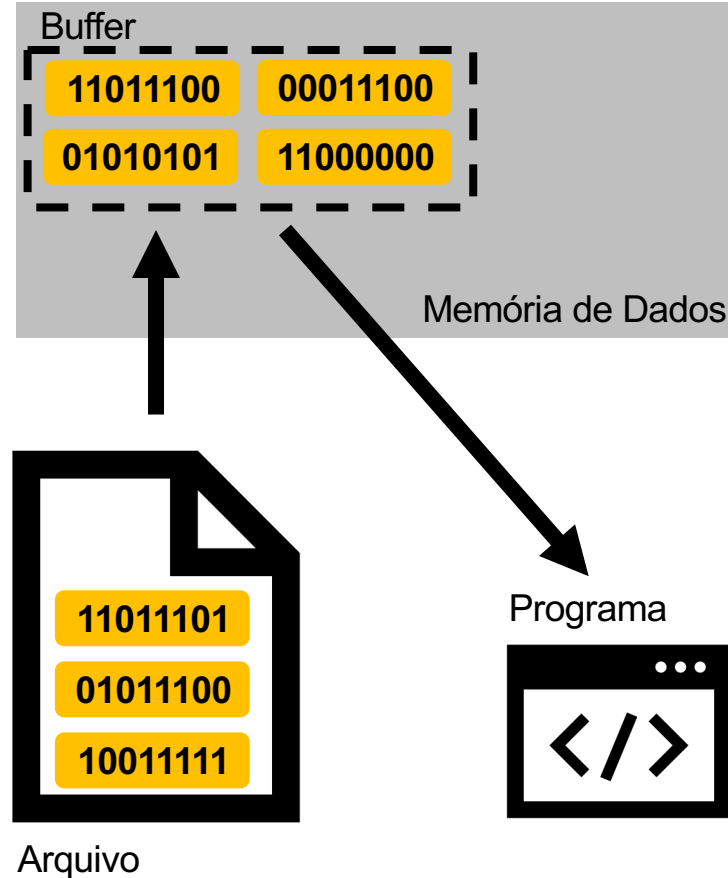
Cursor

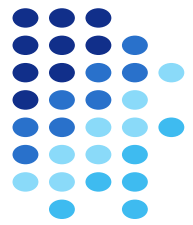
- Um **cursor** é associado ao arquivo de forma a indicar a próxima posição a ser lida ou gravada
- O **cursor** é inicializado com **0** na abertura do arquivo
- O **cursor** é incrementado a cada operação de **leitura** ou **escrita** no arquivo



Buffer

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

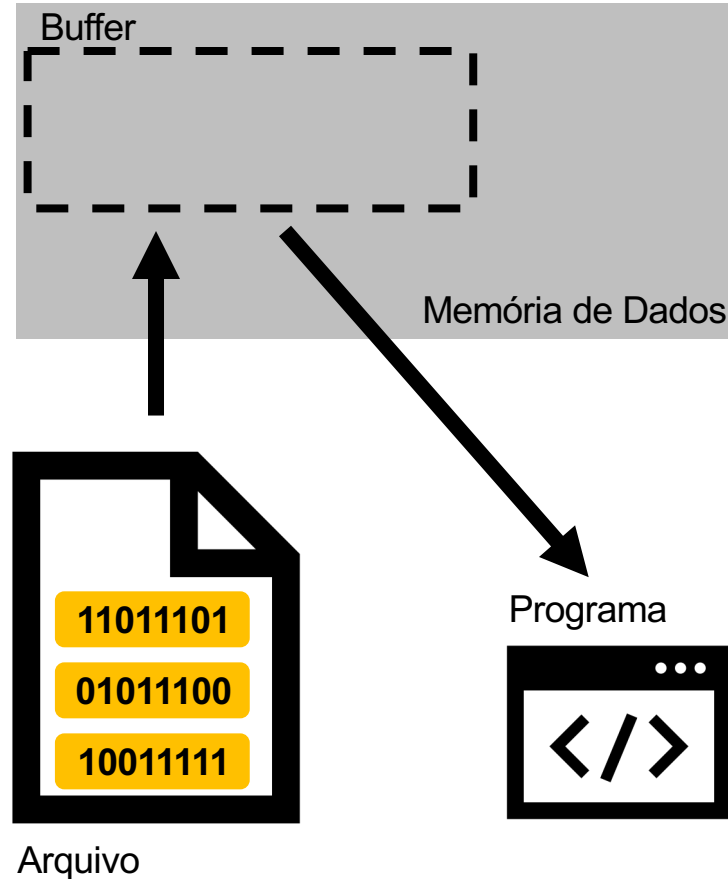


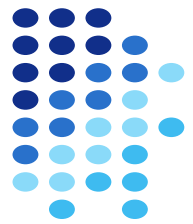


Buffer para Leitura

Situação inicial:

- **buffer vazio**

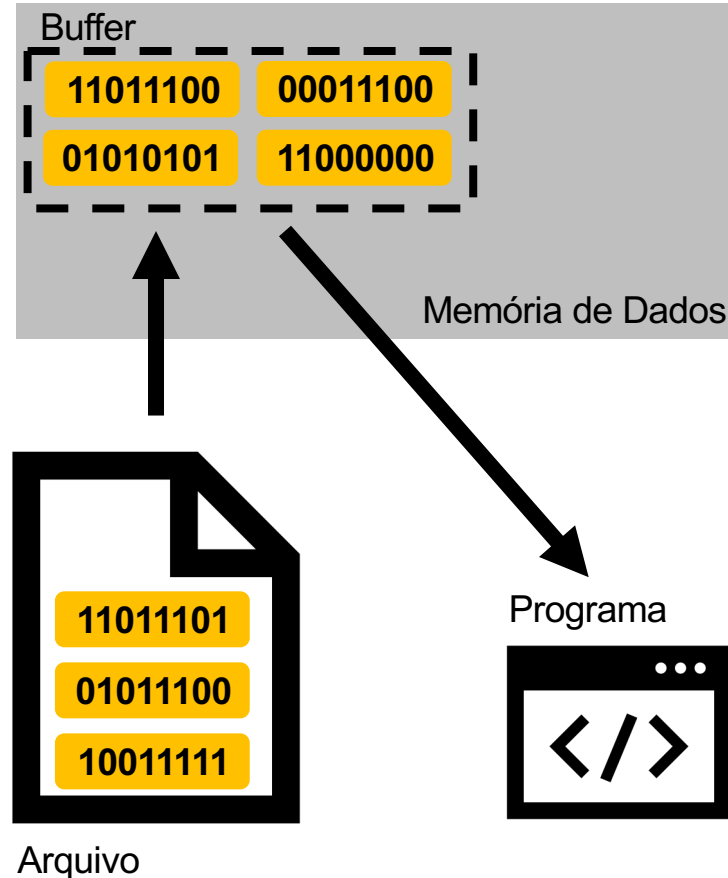


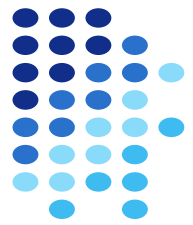


Buffer para Leitura

Programa faz operação de leitura

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

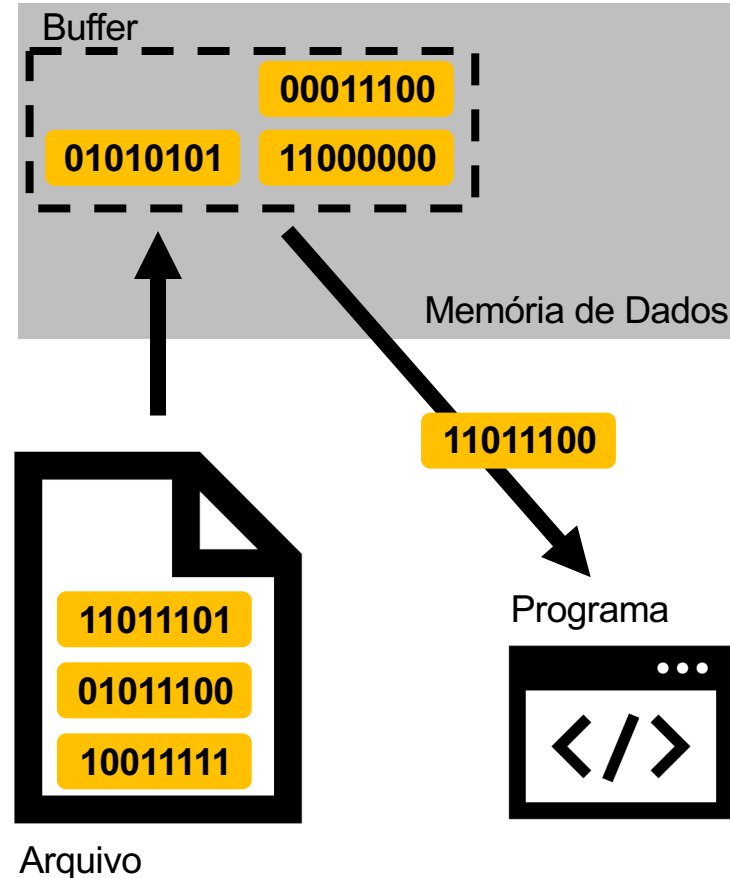


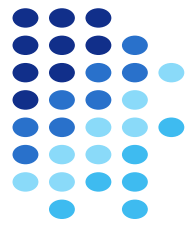


Buffer para Leitura

Programa faz operação de leitura

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

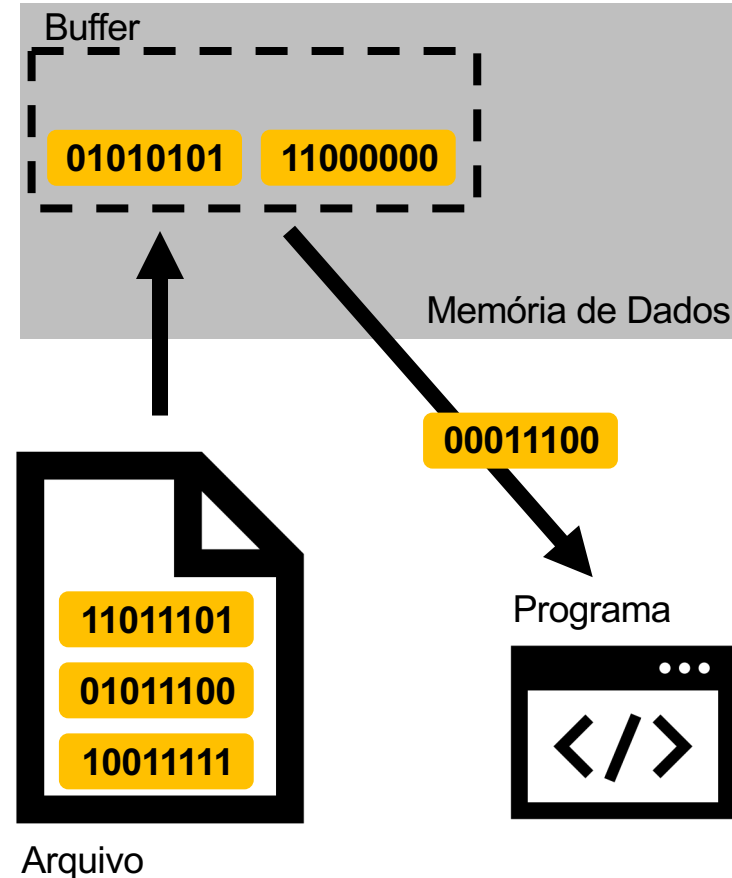


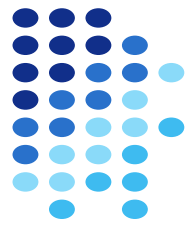


Buffer para Leitura

Programa faz operação de leitura

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

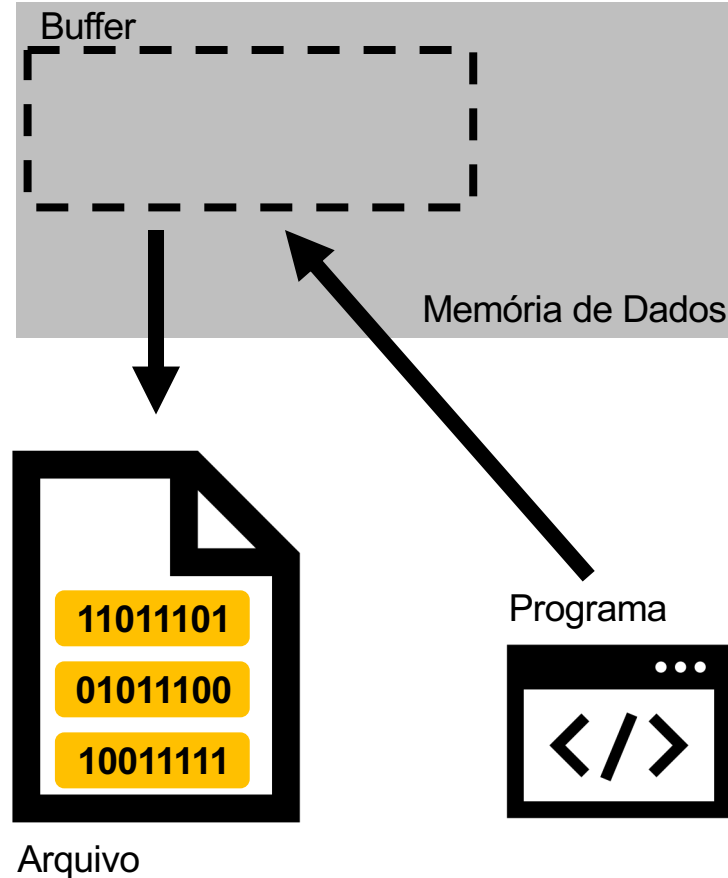


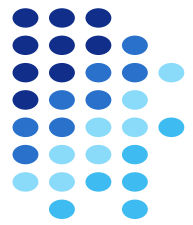


Buffer para Escrita

Situação inicial:

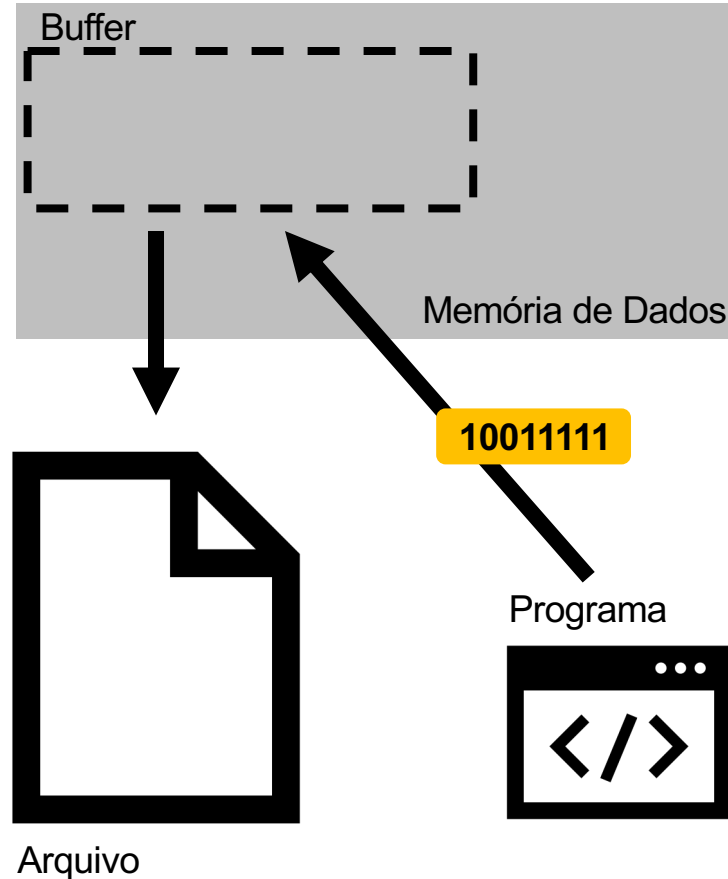
- **buffer vazio**

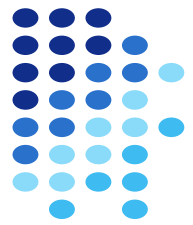




Buffer para Escrita

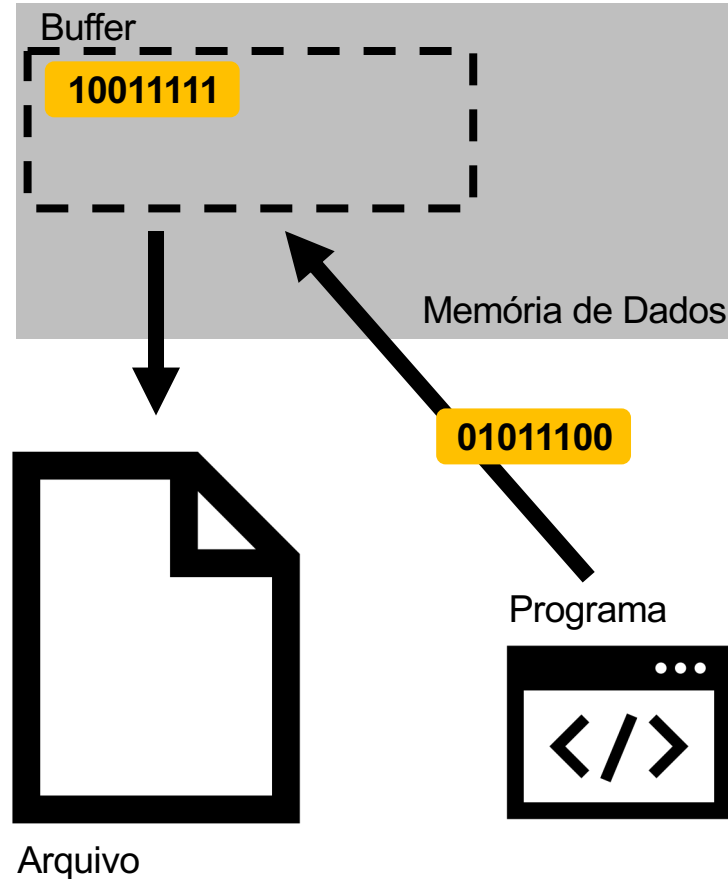
Escrita vai sendo feita no buffer

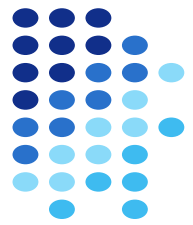




Buffer para Escrita

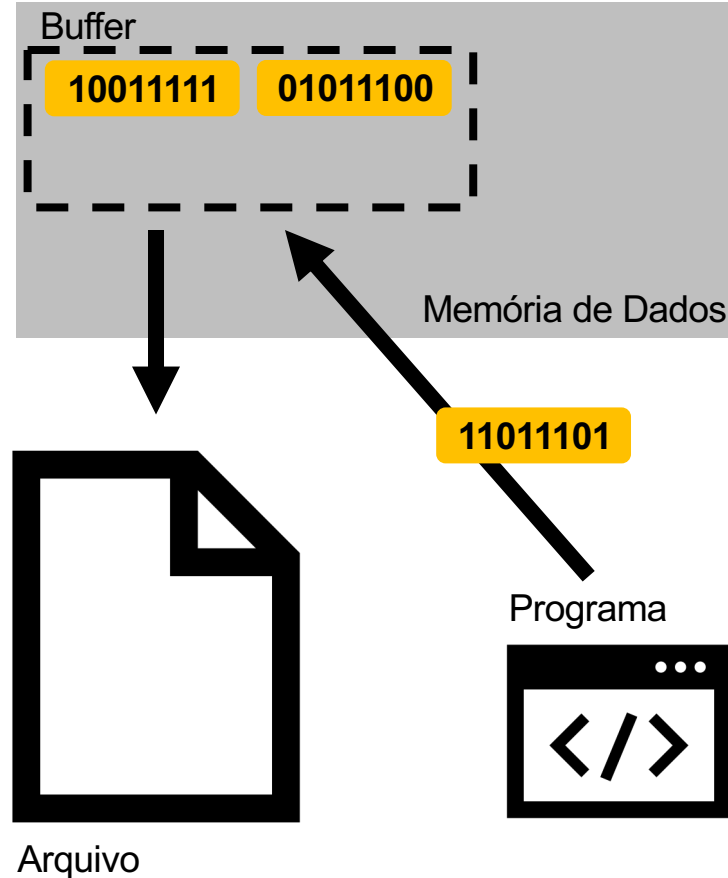
Escrita vai sendo feita no buffer

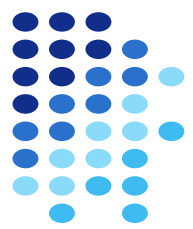




Buffer para Escrita

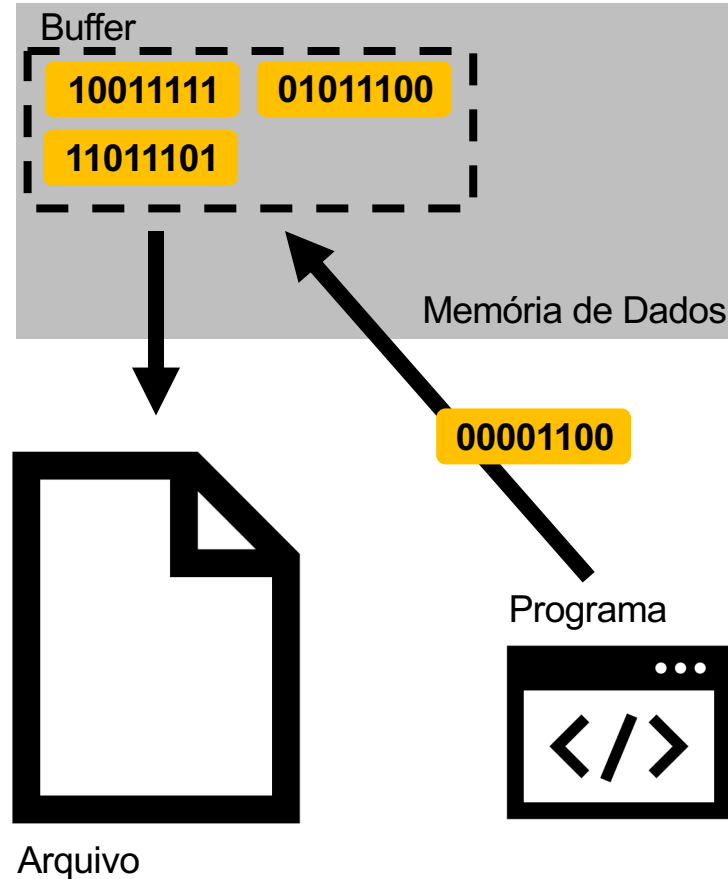
Escrita vai sendo feita no buffer

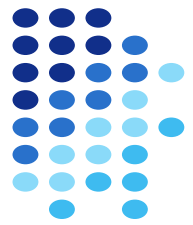




Buffer para Escrita

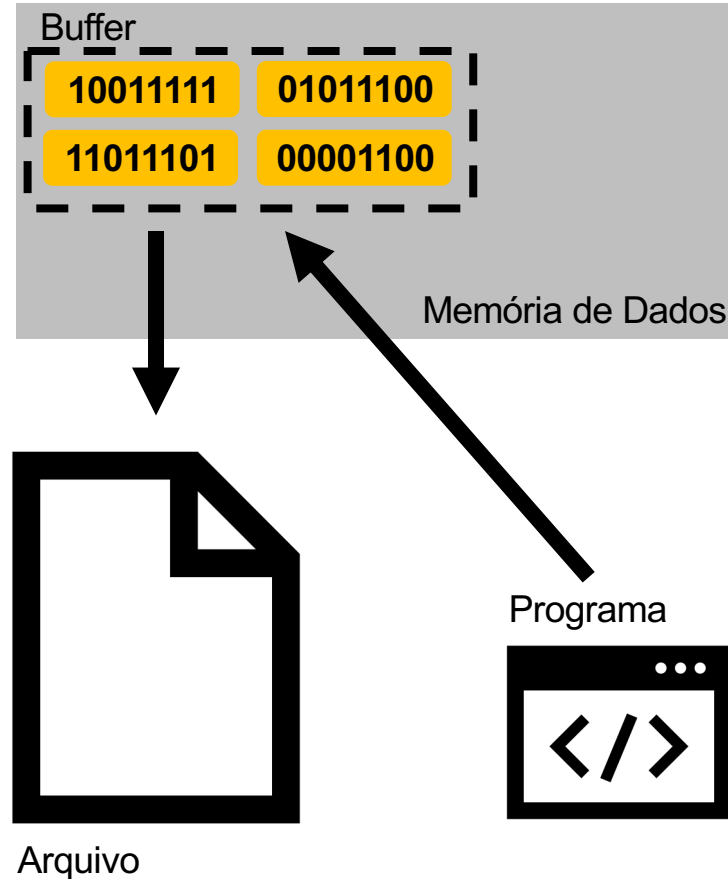
Escrita vai sendo feita no buffer

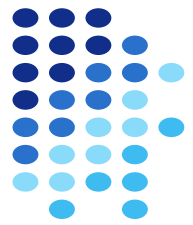




Buffer para Escrita

Quando o buffer enche, ele é automaticamente descarregado no arquivo



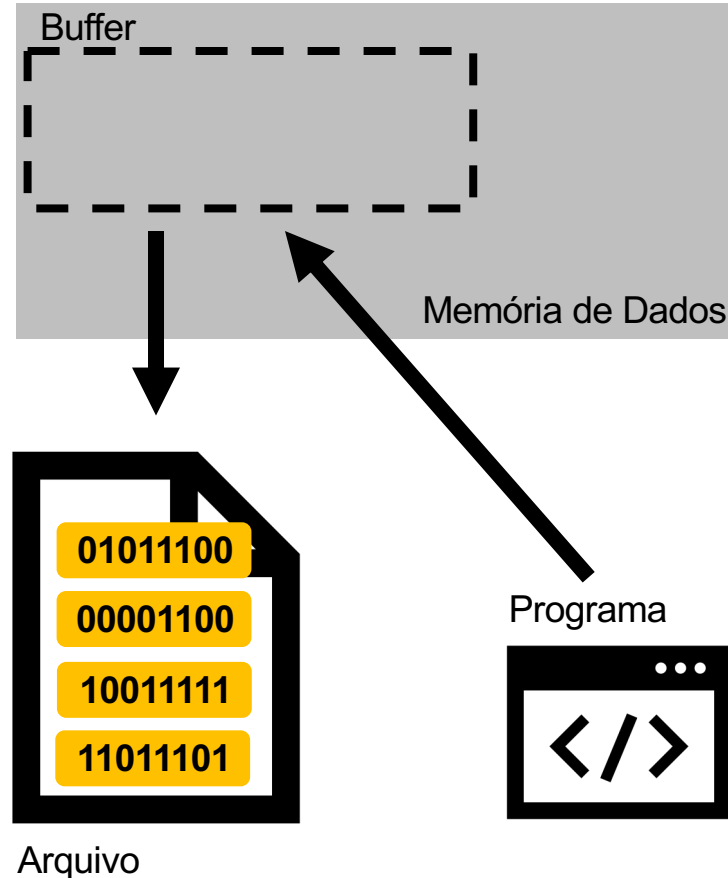


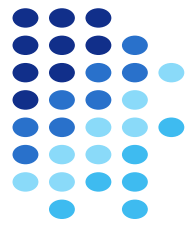
Buffer para Escrita

Quando o buffer enche, ele é automaticamente descarregado no arquivo, ou

O buffer é esvaziado para aguardar novas escritas

Essa operação é chamada de **flush**

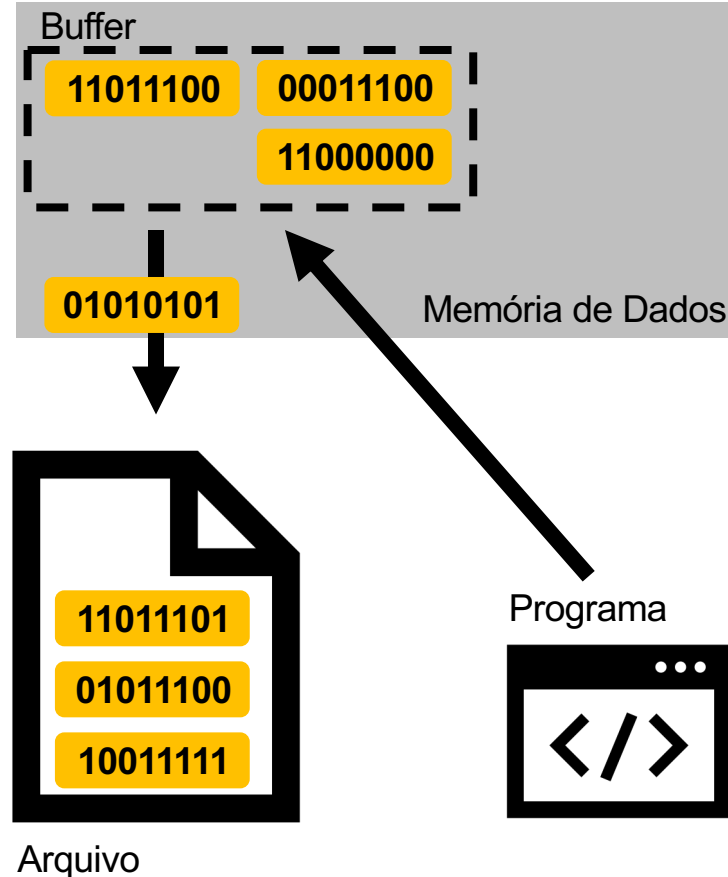


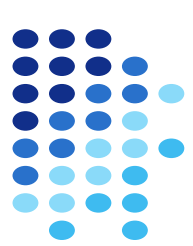


Flush

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

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

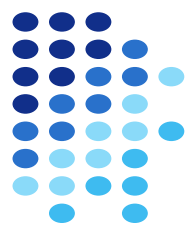




Tipos de Arquivos

Arquivo
Texto

Arquivo
Binário

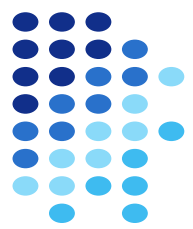


Arquivo Texto

- Conteúdo é apenas **texto**



Imagem: <https://neilpatel.com/br/blog/formatos-de-imagem/>



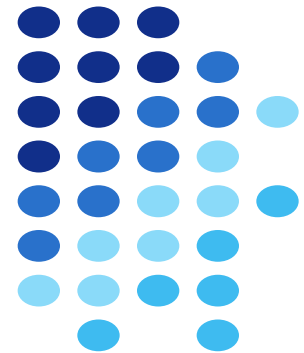
Arquivo Binário

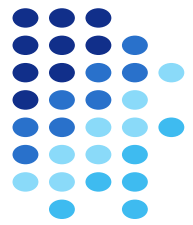
- Conteúdo é **binário**



Imagem: <https://neilpatel.com/br/blog/formatos-de-imagem/>

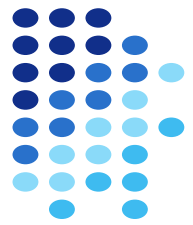
Manipulação de Arquivos em C





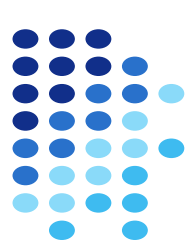
Abertura de Arquivo

- 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**



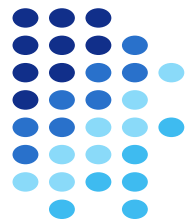
Fechamento de Arquivo

- A operação de fechamento de arquivo **desassocia o arquivo do stream**
- **Libera a memória** (equivale ao **free** para memória alocada dinamicamente)
- Se um arquivo aberto para escrita for fechado, o **conteúdo associado ao seu buffer** 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()	Retorna VERDADEIRO se o fim do arquivo for atingido
ferror()	Retorna VERDADEIRO se ocorreu erro
remove()	Apaga um arquivo
fflush()	Descarrega o buffer no arquivo

Exemplo

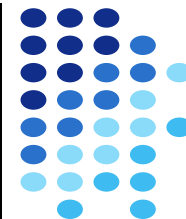


```
#include <stdio.h>
#include <stdlib.h>

int main() {
    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



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

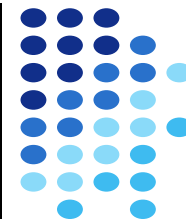
Nome do Arquivo

```
int main() {
    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");
}
```

Modo de abertura
r = leitura de arquivo texto

Exemplo



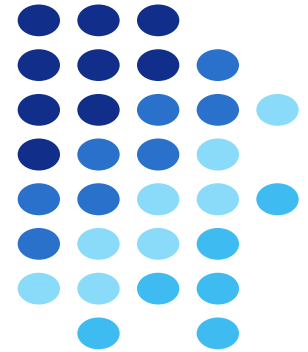
Se houver **erro**
na abertura, **arq**
ficará com o
valor **NULL**

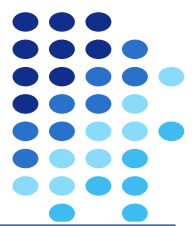
```
#include <stdio.h>
#include <stdlib.h>

int main() {
    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");
}
```

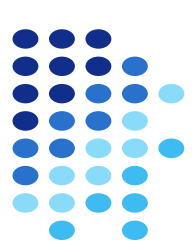

Manipulação de Arquivos Texto em C





Modos de Abertura de Arquivos Texto

Modo	Significado
r	Abre um arquivo texto para <u>leitura</u> (se não existir, retorna NULL)
w	Abre um arquivo texto para <u>escrita</u> (se já existir, conteúdo é apagado, se não existir, será criado)
a	Abre (se já existir) ou cria um arquivo texto para <u>escrita</u> , preservando o conteúdo já existente
r+	Abre um arquivo texto para <u>leitura</u> e <u>escrita</u> (se não existir, retorna NULL)
w+	Cria e abre um arquivo texto para <u>leitura</u> e <u>escrita</u> (se arquivo já existe, conteúdo é apagado)
a+	Abre (se já existir) ou cria um arquivo texto para <u>leitura</u> e <u>escrita</u> – 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 (mesmo que console)

Exemplo de arquivo texto.

Ele 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];
        while (!feof(arq)) { //testa se chegou ao final do
            //arquivo

            fscanf(arq, "%s", s);
            printf("%s\n", s);
        }
        fclose(arq); //fecha arquivo
    }
    else{
        printf("Erro ao abrir arquivo\n");
    }
}
```

dados.txt

Exemplo de arquivo texto.
Ele tem várias linhas:
1
2
3

Console

Exemplo
de
arquivo
texto.
Ele
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];
        while(fscanf(arq, "%s", s) != EOF){
            printf("%s\n", s);
        }
        fclose(arq); //fecha arquivo
    }
    else{
        printf("Erro ao abrir arquivo\n");
    }
}
```

Leitura feita diretamente
na condição do **while**

dados.txt

Exemplo de arquivo texto.
Ele tem várias linhas:
1
2
3

Console

Exemplo
de
arquivo
texto.
Ele
tem
várias
linhas:
1
2
3

Exemplo – Leitura String a String como Números (caso o arquivo só contenha números)

```
void le_arquivo_strings_como_numeros(){
    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;
        while (!feof(arq)) { //testa se chegou ao final do arquivo
            fscanf(arq, "%d", &n);
            printf("%d\n", n);
        }
        fclose(arq); //fecha arquivo
    }
    else
        printf("Erro ao abrir arquivo\n");
}
```

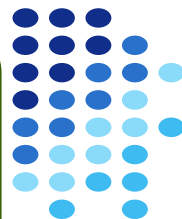
O parâmetro é um ponteiro.
Por isso passamos o endereço (&) do tipo int. No caso de uma **string** (vetor de char) não precisamos passar o endereço, pois o vetor de char já é um ponteiro.

numeros.txt

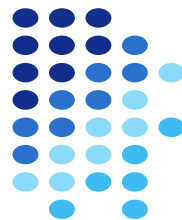
10 20
30
40
50 60 70
80

Console

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");  
}
```

numerosGravados.txt

10
13
16
...
91
94
97

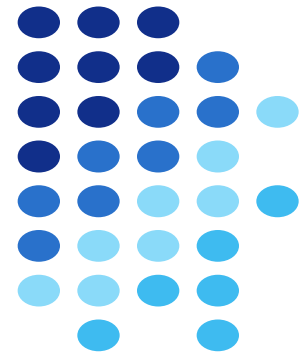
O valor a ser gravado é do tipo int. Assim, usamos %d. Porém, podemos utilizar os mesmos formatos do printf (%s para string, %f para float, etc.)

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**.

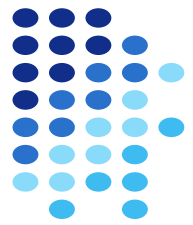
```
void merge(char* nomeArq1, char* nomeArq2, char* nomeArqMerge)
```

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 veículo (**concreto**)
 - Contas-corrente dos clientes de um banco (**abstrato**)
 - Ligações telefônicas dos clientes de uma empresa de telefonia (**abstrato**)

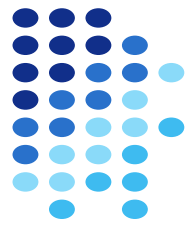


Atributos

- Cada uma dessas entidades pode ser descrita por um conjunto de **atributos**
 - **Funcionário:** nome, CPF, data-nascimento, salário
 - **Carro:** marca, modelo, ano-fabricação, placa
 - **Contas-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,27/07/2002>,<salário,3000>

Carro:

<marca, Renault>, <modelo,Sandero>,<ano-fabricação,2014>,<placa,XYZ0123>

Conta-Corrente:

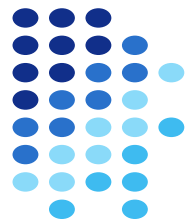
<agencia, 0123>, <conta,123456>,<saldo,2000>

Ligação Telefônica:

<data,01/10/2019>,<origem,79-99000-1234>,<destino,79-99111-4321>,<duração,8'43''>

Tabela

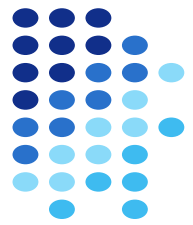
- Uma tabela é um conjunto ordenado de registros. Uma tabela pode ser armazenada em **memória principal** ou em **memória secundária** (disco)
- Nesse segundo caso, também costuma ser chamada de **arquivo**



Exemplo: Arquivo de Funcionários

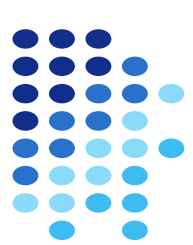
Nome	CPF	Data-Nascimento	Salário
João	012345678-90	10/05/1982	3500
Maria	234567890-12	23/11/1985	2000
Liz	345678901-23	17/09/1950	6000

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



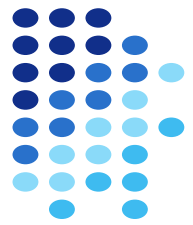
Problema: Encontrar Registros

- Problema comum em diversas aplicações: **encontrar um ou mais registros em uma tabela**
 - Encontrar o empregado de nome Maria
 - Encontrar todos empregados que ganham 3500
 - Encontrar todos os empregados que nasceram em 23/11/1985



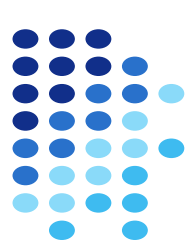
Conceito de Chave

- Dados usados para encontrar um registro: **chave**
- **Chave**: subconjunto de atributos que identifica um determinado registro



Chave Primária e Secundária

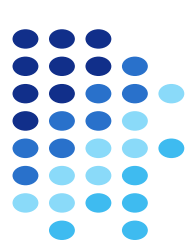
- **Chave Primária:** subconjunto de atributos que identifica unicamente um determinado registro. Exemplo: **CPF do funcionário** ou **RG do funcionário**
 - Na hipótese de uma chave primária ser formada por uma combinação de campos, essa combinação deve ser mínima (não deve conter campos supérfluos)
 - 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**



Tabelas

Aplicações reais lidam com várias tabelas, cada uma delas representando uma entidade

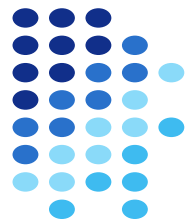
Uma aplicação de controle bancário que precisa controlar clientes, agências e contas-correntes precisaria de quais tabelas?



Tabelas

Uma aplicação de controle bancário que precisaria de quais tabelas?

- Cliente
- Agência
- Conta-Corrente

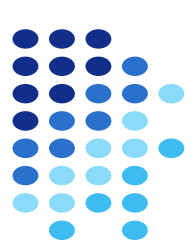


Certa Redundância é Necessária

Neste caso, é necessário correlacionar os dados, para que seja possível saber que conta pertence a que agência, e que conta pertence a que cliente

Para isso, é usual repetir a chave primária da tabela referenciada no outro arquivo

- **Cliente:** **CodCliente**, Nome, CPF, Endereço
- **Agência:** **CodAgencia**, NumeroAgencia, Endereco
- **Conta-Corrente:** **CodAgencia**, **CodCliente**, CodConta, NumeroConta, Saldo



Aplicação Financeira

Chaves Primárias:

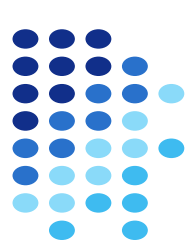
- Cliente: **CodCliente**
- Agência: **CodAgencia**
- Conta-Corrente: **CodAgencia** e **CodCliente**

Chaves Primárias Alternativas:

- Cliente: **CPF**
- Agência: **NumeroAgencia**
- Conta-Corrente: **NumeroConta**

Chaves Secundárias:

- Cliente: **Nome**
- Agência: **Endereço**
- Conta-Corrente: **CodAgencia** ou **CodCliente** ou **Saldo**



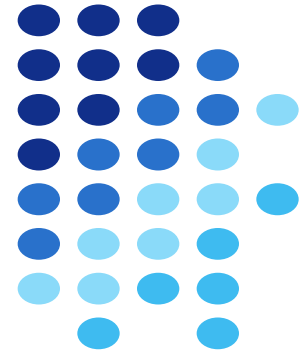
Discussão sobre Chaves

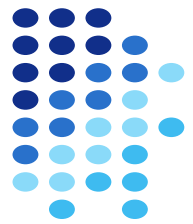
- Por que não usar CPF como chave primária de cliente?
- Por que utilizar os atributos artificiais (código)?

Operações Sobre Arquivos Binários

- Programas que lidam com arquivos realizam os seguintes tipos de operações sobre eles:
 - **Criação:** alocação e inicialização da área de dados, assim como de seus descritores
 - **Destruição:** liberação da área de dados e descritores usados na representação da tabela
 - **Inserção:** inclusão de um novo registro na tabela
 - **Exclusão:** remoção de um registro da tabela
 - **Alteração:** modificação dos valores de um ou mais atributos/campos da tabela
 - **Consulta:** obtenção dos valores de todos os campos de um registro, dada uma chave de entrada

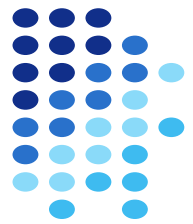
Manipulação de Arquivos Binários em C





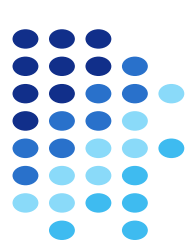
Modos de Abertura de Arquivos Binários

Modo	Significado
rb	Abre um arquivo binário para <u>leitura</u> (se não existir, retorna NULL)
wb	Abre um arquivo binário para <u>escrita</u> (se já existir, conteúdo é apagado, se não existir, será criado)
ab	Abre (se já existir) ou cria um arquivo binário para <u>escrita</u> , preservando o conteúdo já existente
rb+	Abre um arquivo binário para <u>leitura</u> e <u>escrita</u> (se não existir, retorna NULL)
wb+	Cria e abre um arquivo binário para <u>leitura</u> e <u>escrita</u> (se arquivo já existe, conteúdo é apagado)
ab+	Abre (se já existir) ou cria um arquivo binário para <u>leitura</u> e <u>escrita</u> – cursor é posicionado no final do arquivo



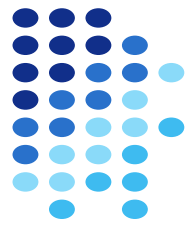
Funções para Manipulação de Arquivos Binários

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()	Retorna verdadeiro se o fim do arquivo for atingido
ferror()	Retorna verdadeiro se ocorreu um erro
remove()	Apaga um arquivo
fflush()	Descarrega um arquivo



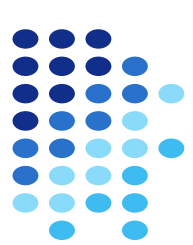
Exemplo

- Gravar registros de funcionários em um arquivo



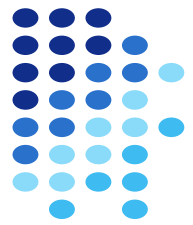
Struct Funcionário

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



Salva Funcionário

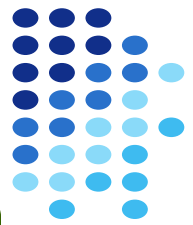
```
// Salva funcionario no arquivo out, na posicao atual do cursor
void salva(TFunc *func, FILE *out) {
    fwrite(&func->cod, sizeof(int), 1, out);
    //func->nome ao invés de &func->nome, pois string já é
    //ponteiro
    fwrite(func->nome, sizeof(char), sizeof(func->nome), out);
    fwrite(func->cpf,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



Salva Funcionário

ORDEM DA GRAVAÇÃO
É importante porque
posteriormente, o
registro deverá ser
lido nessa mesma ordem

```
// Salva funcionario no arquivo out, na posicao 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->salario, sizeof(double), 1, out);
}
```

Salva Funcionário

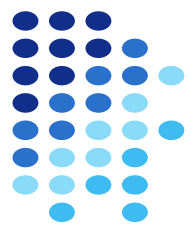
Dado a ser gravado

Arquivo destino

```
// Salva funcionario no arquivo out, na posicao 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->salario, sizeof(double), 1, out);
}
```

Tamanho em bytes do
dado a ser gravado

Número de itens a
serem gravados

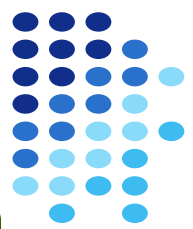


Ler um Registro de Funcionário

```
// Le um funcionario do arquivo in na posicao atual do cursor
// Retorna um ponteiro para funcionario lido do arquivo
TFunc *le(FILE *in) {
    TFunc *func = (TFunc *) malloc(sizeof(TFunc));
    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->salario, sizeof(double), 1, in);
    return func;
}
```

Ler um Registro de Funcionário

```
// Le um funcionario do arquivo in na posicao atual do cursor
// Retorna um ponteiro para funcionario lido do arquivo
TFunc *le(FILE *in) {
    TFunc *func = (TFunc *) malloc(sizeof(TFunc));
    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->salario, sizeof(double), 1, in);
    return func;
}
```



Ler um Registro de Funcionário

```
// Le um funcionario do arquivo
// Retorna um ponteiro para funcionario lido do arquivo
TFunc *le(FILE *in) {
    TFunc *func = (TFunc *) malloc(sizeof(TFunc));
    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->salario, sizeof(double), 1, in);
    return func;
}
```

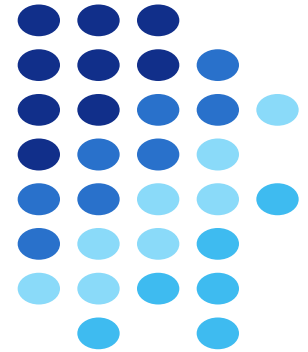
Variável para guardar
dado a ser lido

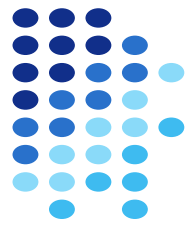
Arquivo fonte

Tamanho em bytes do
dado a ser lido

Número de itens a
serem lidos

Tipos de Acesso à Arquivos





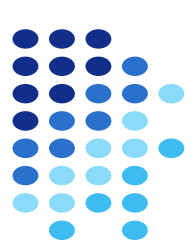
Tipos de Acesso à Arquivos



**Arquivos de
Acesso Sequencial**

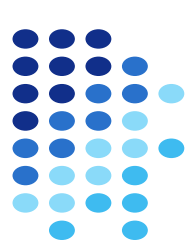


**Arquivos de
Acesso Direto**



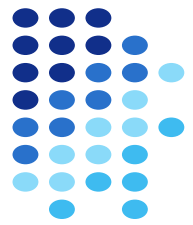
Arquivo de Acesso **Sequencial**

- **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



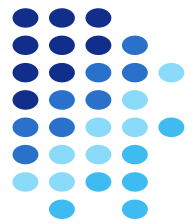
Arquivo de Acesso **Direto**

- São arquivos 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)
- **$\text{EndereçoReg}(i) = (i - 1) * \text{tamanhoReg}$**
- onde **tamanhoReg** é o tamanho dos registros do arquivo, em bytes



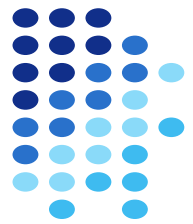
Exemplo

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

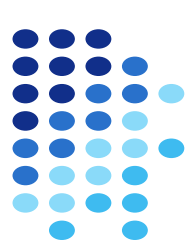
	CodCli	Nome	DataNascimento
0	10	Joao	02/12/1990
26	02	Maria	04/10/1976
52	15	Carlos	30/06/1979
78	04	Carolina	14/05/2000
104	01	Murilo	23/10/1988

Tamanho Registro:

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

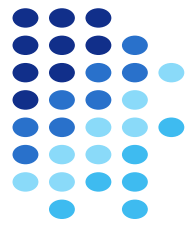
Total: 26 bytes por registro

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



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

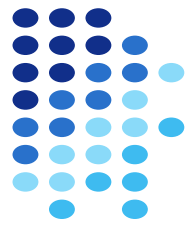


fseek

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

flag pode ser

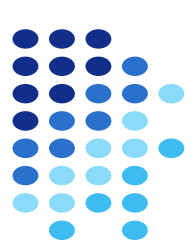
- **seek_set**: indica que deslocamento será feito a partir do início do arquivo
- **seek_cur**: indica que deslocamento será feito a partir da posição atual
- **seek_end**: indica que deslocamento será feito a partir do final do arquivo (útil para descobrir o total de registros em um arquivo, combinado com o uso da função **ftell**, que retorna a posição atual do cursor, em bytes)



Exemplo: Descobrir o Número de Registros de um Arquivo

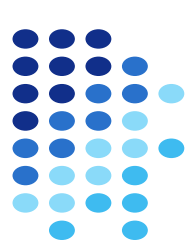
```
/*  
 * tamanho do registro, retornado pela função  
 * tamanho_registro() é dado em bytes  
 */  
int total_registros(FILE *arq) {  
    fseek(arq, 0, SEEK_END);  
    int tam = trunc(ftell(arq) / tamanho_registro());  
    return tam;  
}
```

```
int tamanho_registro() {  
    return sizeof(int) //cod  
        + sizeof(char) * 50 //nome  
        + sizeof(char) * 15 //cpf  
        + sizeof(double); //salario  
}
```



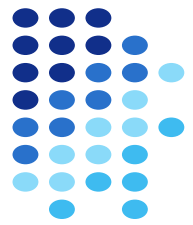
Busca em Arquivo

- Suponha que um banco mantém seus funcionários em um arquivo (mais de 10.000 funcionários)
- O banco deseja dar um aumento para o funcionário de código 305
- Como encontrar o funcionário?




Alternativa 1 – Busca Sequencial

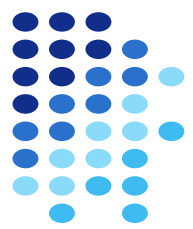
- Ler o arquivo do início até encontrar o funcionário
 - Muito custoso
 - No pior caso (funcionário não existe ou é o último), lê o arquivo inteiro




Busca Sequencial do Funcionário 305



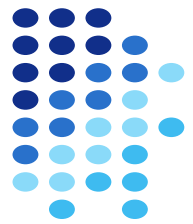
Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500



Busca Sequencial do Funcionário 305



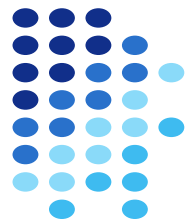
Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500



Busca Sequencial do Funcionário 305

Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500

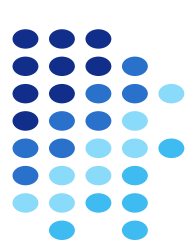




Busca Sequencial do Funcionário 305

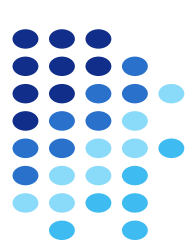
Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500





Busca Sequencial do Funcionário 305

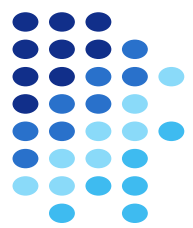
Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
➔ 239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500



Busca Sequencial do Funcionário 305

Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500

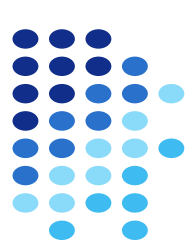




Busca Sequencial do Funcionário 305

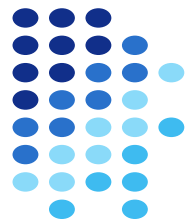
Código	Nome	Salário
102	Joao Silva	1000
123	Carlos Albuquerque	1500
143	Ana Bueno	1500
200	Caio Gusmao	4000
239	Bianca Amarilo	3000
254	Arnaldo Souza	4300
305	Marisa Clara	5000
403	Bruno Simao	4500
410	Guilherme Santos	2000
502	Tatiana Andrade	2500





Alternativa 2 – Busca Binária

- Se o arquivo está ordenado, podemos fazer uma **busca binária**

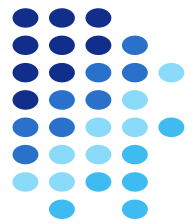


Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500

Lê registro do meio e
compara chave
buscada com a chave
do registro lido

início = 1
fim = 10
meio = trunc((início + fim)/2) = 5
305 > 239

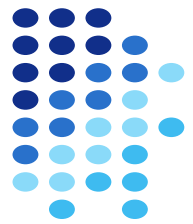


Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500

Repete procedimento
na metade do arquivo
correspondente (se
chave menor, na
metade de cima, se
chave maior, na
metade de baixo)

início = meio + 1

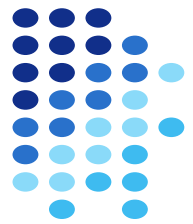


Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500

Lê registro do meio e
compara chave
buscada com a chave
do registro lido

início = 6
fim = 10
meio = trunc((início + fim)/2) = 8
305 < 403

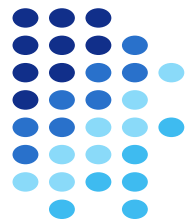


Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500

Repete procedimento
na metade do arquivo
correspondente (se
chave menor, na
metade de cima, se
chave maior, na
metade de baixo)

fim = meio - 1



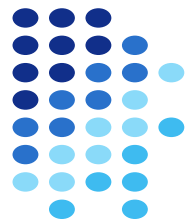
Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500



Lê registro do meio e
compara chave
buscada com a chave
do registro lido

início = 6
fim = 7
meio = trunc((início + fim)/2) = 6
305 > 254

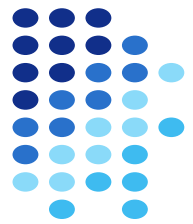


Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
[7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500

Repete procedimento
na metade do arquivo
correspondente (se
chave menor, na
metade de cima, se
chave maior, na
metade de baixo)

início = meio + 1

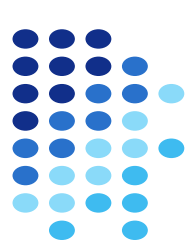


Busca Binária do Funcionário 305

	Código	Nome	Salário
1	102	Joao Silva	1000
2	123	Carlos Albuquerque	1500
3	143	Ana Bueno	1500
4	200	Caio Gusmao	4000
5	239	Bianca Amarilo	3000
6	254	Arnaldo Souza	4300
7	305	Marisa Clara	5000
8	403	Bruno Simao	4500
9	410	Guilherme Santos	2000
10	502	Tatiana Andrade	2500

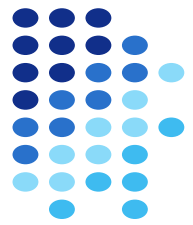
Lê registro do meio e
compara chave
buscada com a chave
do registro lido

início = 7
fim = 7
meio = $\text{trunc}((\text{início} + \text{fim})/2) = 7$
305 = 305



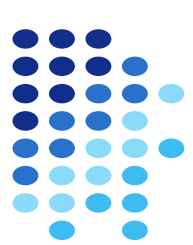
Busca Binária - Detalhes

- Exige que se saiba o endereço de um determinado registro, para que seja possível fazer o **seek** no arquivo para aquele endereço
 - Usar cálculo de endereço, visto anteriormente
- Exige que se saiba quantos registros o arquivo possui
 - Usar função **total_registros()**, vista anteriormente



Comparação

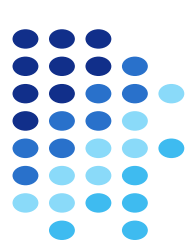
- Na busca sequencial, para esse exemplo, foram lidos 7 registros até encontrar o funcionário desejado
- Na busca binária, foram lidos apenas 4 registros
- Assumindo que o arquivo tem **n** registros:
 - Complexidade da busca sequencial: **$O(n)$**
 - Complexidade da busca binária: **$O(\log n)$**



Exercício

- Dado um arquivo de funcionários, ordenado, implementar uma função que faz busca binária no arquivo de forma recursiva

```
/*      cod é a chave buscada
 *      *arq é o ponteiro para o arquivo
 *      inicio é a posicao inicial, indice inicial
 *      fim é a posicao final, indice final
 */
TFunc *buscaBinaria(int cod, FILE *arq, int inicio, int fim)
```



Referências

- Material baseado nos slides de **Vanessa Braganholo**, Disciplina de Estruturas de Dados e Seus Algoritmos. Instituto de Computação. Universidade Federal Fluminense (UFF), Niterói, Brasil.
- Schildt, H. C Completo e Total. Ed. McGraw-Hill.