

Introdução à Ciência da Computação

Disciplina: 113913

Prof. Luiz Augusto Fontes Laranjeira
E-mail: **luiz.laranjeira@gmail.com**

Universidade de Brasília – UnB
Campus Gama

15. ARQUIVOS

Arquivos

- Os comandos de entrada e saída que usamos até o momento foram:
 - **printf** – mostra dados formatados na tela
 - **scanf** – lê dados formatados digitados no teclado
 - **gets** _ lê uma string (que pode conter espaços) digitada no teclado
 - **puts** _ escreve uma string na tela
 - **getchar** _ lê um caracter digitado no teclado e a ecoa na tela
 - **putchar** _ escreve um caracter na tela
- Todavia, dados podem ser lidos e gravados em arquivos (em geral em discos). Um arquivo em disco é um espaço com diversas características físicas, dependendo do tipo de dispositivo que é usado (disquete, disco rígido, CD, etc.). Porém, um arquivo em disco, pode ser visto como um espaço sequencial (“stream”) de onde são lidos ou onde são gravados os dados. A cada arquivo está associado um nome, pelo qual o mesmo é conhecido externamente, isto é, o nome que consta no diretório do disco.
- Uma vez que um arquivo é uma sequência de bytes temos o marcador do final desse arquivo que é: **EOF (EndOfFile)** (valor usado = -1)

(Não se utilizam os caracteres ASCII: EOT = 0x04 ou FS=0x1C)

Arquivos

- Existem dois tipos de arquivos:
- **TEXTO:**
 - Em que são gravados caracteres, ou seja um texto mesmo, semelhante a um programa em C. O arquivo texto pode ser facilmente visualizado com um editor de texto.
- **BINÁRIO:**
 - Em que são gravados os dados como estariam na memória. Por exemplo, uma variável inteira é gravada com 4 bytes com o conteúdo exato que está na memória. Não conseguimos visualizar um arquivo binário simplesmente com um editor de texto. É necessário um programa especial para abrí-lo e fazê-lo ser mostrado na tela.



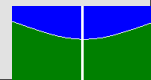
Trabalhando com Arquivos

- Para tratar de arquivos a linguagem C fornece um nível de abstração entre o programador e o dispositivo que estiver sendo usado. Esta abstração é chamada fila de bytes e o dispositivo normalmente é o arquivo. Existe um sistema bufferizado de acesso ao arquivo, onde um **ponteiro de arquivo** define vários aspectos do arquivo, como nome, status e posição corrente, além de ter a fila associada a ele.

Assim criamos um:

```
FILE *fp;
```

Ou seja, um ponteiro fp, do tipo ponteiro de arquivo.



Trabalhando com Arquivos

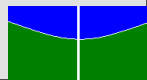
- Antes de ler ou gravar em algum arquivo precisamos ter certeza de algumas coisas:
 - Onde se encontra o arquivo?
 - O arquivo já existe e vamos abri-lo somente para leitura?
 - O arquivo já existe e vamos abri-lo somente para gravação (escrita)?
 - O arquivo já existe e vamos abri-lo para leitura e gravação (escrita)?
 - O arquivo não existe e vamos criá-lo?
 - O arquivo já existe mas vamos recriá-lo (deletando o seu conteúdo prévio)?
- Assim, temos a função **fopen** (file Open), que possui dois parâmetros: um contém o nome do arquivo, e o segundo representa o tipo de operação que faremos com ele.
- Por exemplo:

```
FILE *fp;  
char nomeArquivo[] = "c:\\MeuArquivo.txt";  
fp = fopen(nomeArquivo, "w");
```

Trabalhando com Arquivos

- Caso a função **fopen** não encontre o arquivo indicado, ela retorna NULL. Dessa forma podemos garantir, caso o arquivo já exista, que ele não será recriado (apagado) fazendo o seguinte teste:
- Por exemplo:

```
FILE *fp;  
char nomeArquivo[] = "c:\\MeuArquivo.txt";  
fp = fopen(nomeArquivo, "r+");  
if (fp == NULL) {  
    fp = fopen(nomeArquivo, "w+");  
}
```



Trabalhando com Arquivos

■ Tabela de funcionalidades:

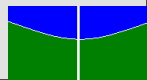
- r : abre um arquivo TEXTO para leitura.
- w: cria um arquivo TEXTO para gravação, ou se o arquivo já existe, elimina seu conteúdo e recomeça a gravação a partir do seu início.
- a: abre um arquivo TEXTO já existente para gravação, a partir de seu final.
- rb: abre um arquivo BINÁRIO para leitura.
- wb: cria um arquivo BINÁRIO para gravação, ou se o arquivo já existe, elimina seu conteúdo e recomeça a gravação a partir do seu início.
- ab: abre um arquivo BINÁRIO já existente para gravação, a partir de seu final.
- 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.



Trabalhando com Arquivos

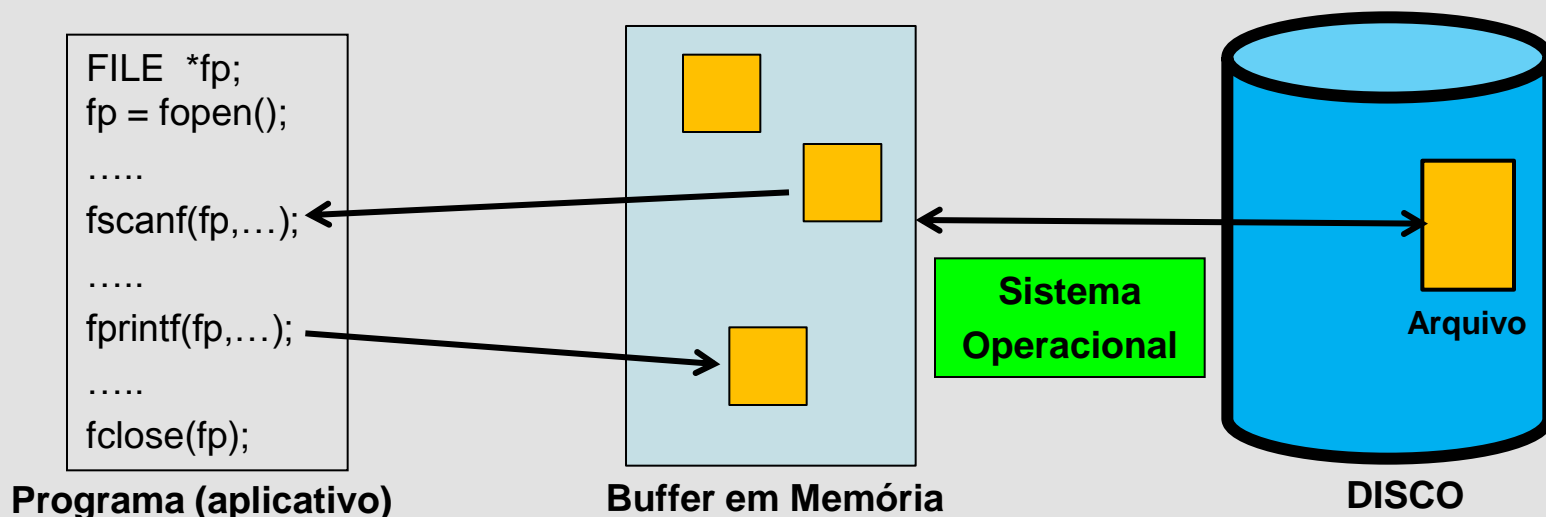
■ Tabela de funcionalidades:

- 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 em que o arquivo não existe.
- 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.



Trabalhando com Arquivos

- Da mesma forma que devemos abrir um arquivo utilizando a função **fopen**, devemos fechá-lo quando não formos mais utilizá-lo, pois assim realmente garantimos que o arquivo será salvo em disco, e não ficará simplesmente no buffer (região de memória).
- Para isso utilizamos o comando **fclose**, da seguinte forma:
`fclose (fp); /* onde fp é o ponteiro para o arquivo */`



Trabalhando com Arquivos

- Para fazermos a leitura e gravação em um arquivo, podemos utilizar as funções ***fprintf***, e ***fscanf***, respectivamente.
- Essas funções tem como parâmetros, o ponteiro do arquivo, os tipos dos dados, e depois, os endereços das variáveis que utilizaremos para representar esses valores.
 - Por exemplo, se queremos ler números inteiros de um arquivo:

```
int numero1, numero2, numero3;  
fscanf(fp, "%d %d %d", &numero1, &numero2, &numero3);
```

- Para escrever no arquivo:

```
fprintf(fp, "%d %d %d", &numero1, &numero2, &numero3);
```

Trabalhando com Arquivos

- Para fazermos a leitura e gravação em um arquivo, podemos também utilizar as funções ***fputs***, e ***fgets***, respectivamente.
- Essas funções tem como parâmetros, o ponteiro do arquivo, o endereço do vetor/buffer de onde se quer ler uma string ou onde se quer escrever uma string.
 - Por exemplo, se queremos ler uma string de um arquivo:

```
char string[80];  
fgets(string, 80, fp);
```

- Para escrever no arquivo:

```
fputs(string, fp);
```

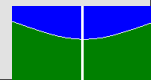


Trabalhando com Arquivos

- Uma vez que utilizamos um ponteiro para ler cada caracter ou byte (valor binário) de um arquivo, ou para escrever no arquivo, e que este ponteiro é incrementado quando lemos ou escrevemos, precisamos muitas vezes reposicionar o ponteiro no inicio do arquivo. Para isso podemos utilizar a função ***rewind***, da seguinte forma:

```
rewind (FILE *fp);
```

```
/* retorna a posição corrente do arquivo para o início */
```



Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo, em seguida lê a mesma (do arquivo) e mostra seu conteúdo na tela:

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

```
int main () {
```

```
    FILE *fp;
```

```
    char string[50];
```

```
    char nomeArquivo[50];
```

```
    strcpy(nomeArquivo, "MeuArquivo.txt");
```

```
    if (fopen(nomeArquivo, "r+") == NULL) {
```

```
        fp = fopen(nomeArquivo, "w");
```

```
    }
```

```
    strcpy(string, "Meu primeiro programa com arquivo texto");
```

```
    fprintf(fp, "%s", string);
```

```
    fclose(fp);
```

```
    fopen(nomeArquivo, "r+");
```

```
    strcpy(string, " "); /*mudamos o conteudo da variavel string*/
```

```
    fscanf(fp, "%s", string);
```

```
    printf("String contida no arquivo: %s\n", string);
```

```
    getchar();
```

```
    fclose(fp);
```

```
    return 0;
```

```
}
```

Tenta abrir o arquivo para leitura e gravação (testando se ele existe)

Cria o arquivo e abre para gravação

Grava no arquivo

Fecha o arquivo

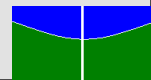
Lê do arquivo

Abre o arquivo

Fecha o arquivo

Arquivo Texto

- OBS sobre o exemplo do slide anterior:
 - Note que a saída resultante é simplesmente a palavra *Meu*, pois num arquivo do tipo texto, os espaços indicam um novo registro (dado).
 - Para solucionar isso, temos tres alternativas:
 1. Fazer um loop para ler as strings do arquivo enquanto não chegar no final do mesmo (veja exemplo no próximo slide)
 2. Utilizar o comando `fscanf` da seguinte forma:
`fscanf(fp, "%[^\n]s", string);` (ver exemplo dois slides adiante);
 3. Utilizar o comando `fgets` da seguinte forma:
`fgets(string, 50, fp);`
 - Observe que podemos abrir o arquivo `MeuArquivo.txt` usando o NotePad e é possível visualizar o conteúdo que foi escrito nele.



fgets

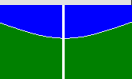
```
char * fgets ( char * str, int num, FILE * stream );
```

Get string from stream

Reads characters from *stream* and stores them as a C string into *str* until (*num-1*) characters have been read or either a newline or a the End-of-File is reached, whichever comes first.

A newline character makes fgets stop reading, but it is considered a valid character and therefore it is included in the string copied to *str*.

A null character is automatically appended in *str* after the characters read to signal the end of the C string.



fgets (cont.)

char * fgets (char * str, int num, FILE * stream);

Parameters

- *str* Pointer to an array of chars where the string read is stored.
- *num* Maximum number of characters to be read (including the final null-character). Usually, the length of the array passed as *str* is used.
- *stream* Pointer to a FILE object that identifies the stream where characters are read from.
To read from the standard input, *stdin* can be used for this parameter.



fgets (cont.)

```
char * fgets ( char * str, int num, FILE * stream );
```

Return Value

- On success, the function returns the same *str* parameter.
- If the EOF (End-of-File) is encountered and no characters have been read, the contents of *str* remain unchanged and a null pointer is returned.
- If an error occurs, a null pointer is returned.
- Use either **feof** or **ferror** to check whether an error happened or the End-of-File was reached.



fputs

```
int fputs ( const char * str, FILE * stream );
```

Write string to stream

Writes the string pointed by *str* to the *stream*.

The function begins copying from the address specified (*str*) until it reaches the terminating null character ('\0'). This final null-character is not copied to the stream.

Parameters

- *str* An array containing the null-terminated sequence of characters to be written.
- *stream* Pointer to a FILE object that identifies the stream where the string is to be written.

Return Value

- On success, a non-negative value is returned.
- On error, the function returns EOF.



Arquivo Texto

Exemplo ARQUIVO TEXTO: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo, em seguida lê a mesma (do arquivo) e mostra seu conteúdo na tela:

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

int main () {
    FILE *fp;
    char string[50];
    char nomeArquivo[50];
    strcpy(nomeArquivo, "MeuArquivo.txt");
    if (fopen(nomeArquivo, "r+") == NULL) {
        fp = fopen(nomeArquivo, "w");
    }

    strcpy(string, "Meu primeiro programa com arquivo texto");
    fprintf(fp, "%s", string);
    fclose(fp);
    fopen(nomeArquivo, "r+");
    strcpy(string, ""); /*mudamos o conteudo da variavel string*/
    while (fscanf(fp, "%s", string) != EOF) {
        printf("%s ", string);
    }
    fclose(fp);
    return 0;
}
```

Uma vez que fechamos o arquivo e o reabrimos não há necessidade de usarmos o `rewind()` para o ponteiro voltar a posicao original, pois quando abrimos o arquivo ele já está lá.

Note que cada vez que chamamos a funcao *fscanf* ela incrementa automaticamente o ponteiro do arquivo

Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo, em seguida lê a mesma (do arquivo) e mostra seu conteúdo na tela:

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

int main () {
    FILE *fp;
    char string[50];
    char nomeArquivo[50];
    strcpy(nomeArquivo, "MeuArquivo.txt");
    if (fopen(nomeArquivo, "r+") == NULL) {
        fp = fopen(nomeArquivo, "w");
    }
    strcpy(string, "Meu primeiro programa com arquivo texto");
    fprintf(fp, "%s", string);
    fclose(fp);
    fopen(nomeArquivo, "r+");
    strcpy(string, " "); /*mudamos o conteudo da variavel string*/
    fscanf(fp, "%[^\n]s", string);
    printf("String contida no arquivo: %s\n", string);
    getchar()
    fclose(fp);
    return 0;
}
```

Lê toda a string de uma vez.

Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo, em seguida lê a mesma (do arquivo) e mostra seu conteúdo na tela:

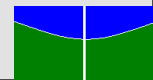
```
#include <stdio.h>
#include <string.h>

int main () {
    FILE *fp;
    char string[50];
    char nomeArquivo[50];
    strcpy(nomeArquivo, "MeuArquivo.txt");
    if (fopen(nomeArquivo, "r+") == NULL) {
        fp = fopen(nomeArquivo, "w");
    }
    strcpy(string, "Meu primeiro programa com arquivo texto");
    fputs(string, fp);
    fclose(fp);
    fopen(nomeArquivo, "r+");
    strcpy(string, " "); /*mudamos o conteudo da variavel string*/
    fgets(string, 50, fp);
    printf("String contida no arquivo: %s\n", string);
    getchar();
    fclose(fp);
    return 0;
}
```

Lê toda a string de uma vez.

Arquivos Binários

- **ARQUIVOS BINÁRIOS:** São arquivos versáteis, pois possuem maior facilidade de leitura e escrita, pois podemos gravar estruturas inteiras de uma vez, como vetores por exemplo, e acessá-los facilmente, ao contrário dos arquivos TEXTO que temos que escrever (e ler) dado por dado.
- Em arquivos binários usamos basicamente as funções ***fwrite()*** para escrever e ***fread()*** para ler.



Trabalhando com Arquivos

■ ARQUIVOS BINÁRIOS

fwrite(endereço, sizeof(tipo_variavel), qtde a ser gravada, FILE *fp)

fread (endereço, sizeof(tipo_variavel), qtde a ser lida, FILE *fp)

```
struct Dados_t meusdados;
```

```
struct Dados_t vetdados[20];
```

```
char buffer[1000];
```

```
fwrite (&meusdados, sizeof(struct Dados_t), 1, FILE *fp);
```

```
fwrite (vetdados, sizeof(Dados_t), 4, FILE *fp);
```

```
fwrite(buffer, 1, 50, FILE *fp);
```

```
fread (&meusdados, sizeof(struct Dados_t), 1, FILE *fp);
```

```
fread (vetdados, sizeof(struct Dados_t), 3, FILE *fp);
```

```
fread (buffer, 1, 44, FILE *fp);
```

- O endereço, no caso da função **fwrite** é da variável/vetor/buffer onde o(s) dado(s) a ser (em) gravado(s) se encontra(m) e que será(ão) passado(s) para o arquivo. No caso da função **fread** é o endereço da variável /vetor/buffer que irá receber o valor a ser lido do arquivo.

Arquivo Binário – Exemplo 1

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo binário” em um arquivo, em seguida lê a mesma (do arquivo) e mostra seu conteúdo na tela:

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

int main () {
    FILE *fp;
    char string[50];
    char nomeArquivo[50];

    strcpy(nomeArquivo, "MeuArquivoBinario.bin");
    if (fopen(nomeArquivo, "a+b") == NULL) {
        fp = fopen(nomeArquivo, "wb");
    }
    strcpy(string, "Meu primeiro programa com arquivo binario");
    fwrite(string, sizeof(string), 1, fp);
    fclose(fp);
    fp = fopen(nomeArquivo, "a+b");
    strcpy(string, " ");
    fread(string, sizeof(string), 1, fp);
    printf("\nVeja o conteudo da string lida do arquivo: ");
    printf("\n%s ", string);
    getchar();
    fclose(fp);
    return 0;
}
```

Note que foi possível escrever toda a string sem o WHILE. Isso foi possível pois foi guardada no arquivo a estrutura binária da string.

Arquivo Binário – Exemplo 2

Exemplo: um programa que escreve uma STRUCT em um arquivo, e depois lê esta informação do arquivo e mostra na tela. Neste exemplo são lidas e gravadas informações (código e nome) para 3 pessoas.

```
#include <stdio.h>

typedef struct {
    int codigo;
    char nome[30];
} DadosPessoa_t;

int main() {
    FILE *fp;
    DadosPessoa_t dadosDePessoa;
    int i;
    char nomeArquivo[50];

    strcpy(nomeArquivo, "arqbin.bin");
    if (fopen(nomeArquivo, "a+b") == NULL) {
        fp = fopen(nomeArquivo, "wb");
    }

    /* CONTINUA NO PRÓXIMO SLIDE */
}
```

Arquivo Binário – Exemplo 2

/* CONTINUAÇÃO DO SLIDE ANTERIOR */

```
for (i = 0; i < 3; i++) {  
    printf("Informe o nome: ");  
    scanf("%s", dadosDePessoa.nome);  
    printf("Informe o codigo: ");  
    scanf("%d", &dadosDePessoa.codigo);  
    fwrite(&dadosDePessoa, sizeof(DadosPessoa_t), 1, fp);  
}  
fclose(fp);  
fopen(nomeArquivo, "a+b");  
while (fread(&dadosDePessoa, sizeof(DadosPessoa_t), 1, fp) != 0)  
{  
    printf("\n\nNome: %s", dadosDePessoa.nome);  
    printf("\nCodigo: %d", dadosDePessoa.codigo);  
    getchar();  
}  
fclose(fp);  
}
```

Lê uma estrutura e grava no arquivo.



Lê uma estrutura de cada vez do arquivo.



O loop encerra quando a função **fread** retornar 0 (zero) caracteres lidos, ou seja, quando chegar ao fim do arquivo.

Arquivo Binário – Exemplo 3

Exemplo: um programa que escreve um vetor de estruturas (struct) num arquivo, e depois lê esta informação do arquivo e a mostra na tela. Neste exemplo são lidas e gravadas informações (código e nome) para 3 pessoas.

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

```
typedef struct {  
    int codigo;  
    char nome[30];  
} DadosDeFuncionario_t;
```

```
int main() {  
    FILE *fp;  
    DadosDeFuncionario_t dadosDeFuncionario[3], dadoslidos[3];  
    int i;  
    char nomeArquivo[50];  
  
    strcpy(nomeArquivo, "arqbin1.bin");  
    if (fopen(nomeArquivo, "a+b") == NULL) {  
        fp = fopen(nomeArquivo, "wb");  
    }  
}
```

```
/* CONTINUA NO PRÓXIMO SLIDE */
```

Arquivo Binário – Exemplo 3


/* CONTINUAÇÃO DO SLIDE ANTERIOR */

```
for (i = 0; i<3; i++ ) {  
    printf("Informe o nome: ");  
    scanf("%s", dadosDeFuncionario[i].nome);  
    printf("Informe o cod: ");  
    scanf("%d", &dadosDeFuncionario[i].codigo);  
}  
fwrite(dadosDeFuncionario,sizeof(DadosDeFuncionario_t),3,fp);  
fclose(fp);  
fp = fopen(nomeArquivo,"a+b");  
fread(&dadoslidos, sizeof(DadosDeFuncionario_t),3,fp);  
for (i = 0; i<3; i++ ) {  
    printf("\n\nNome: %s",dadoslidos[i].nome);  
    printf("\nCod: %d",dadoslidos[i].codigo);  
    getchar();  
}  
}
```

Grava no arquivo de uma vez só todo o vetor com 3 estruturas.



Lê 3 estruturas de uma vez do arquivo, e escreve as 3 no vetor dadoslidos.



Arquivo Binário

- Note que a manipulação de arquivos binários é de um modo geral simples, todavia se tentarmos abrir o arquivoBinario.bin no notepad veremos que não conseguiremos lê-lo facilmente!



Arquivos

ATENÇÃO:

- Toda vez que estamos trabalhando com arquivos, há uma espécie de **posição atual** no arquivo. Esta é a posição de onde será lido ou escrito o próximo caractere.
- Normalmente, num acesso sequencial a um arquivo, não temos que mexer nesta posição pois quando lemos um caractere a posição no arquivo é automaticamente atualizada. Num acesso randômico teremos que mexer nesta posição (ver função `fseek()` a seguir).

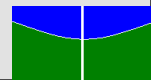


Trabalhando com Arquivos

- ***fseek()***: para se fazer procuras e acessos randômicos em arquivos usa-se a função ***fseek()***. Esta função move a posição corrente de leitura ou escrita no arquivo de um valor especificado, a partir de um ponto especificado.

fseek (FILE *fp, deslocamento em bytes, origem);

- O parâmetro *origem* determina a partir de onde o *deslocamento em bytes* de movimentação serão contados. Os valores possíveis são definidos por macros em **stdio.h** e são:
 - SEEK_SET = 0 = Início do arquivo
 - SEEK_CUR = 1 = Ponto corrente no arquivo
 - SEEK_END = 2 = Fim do arquivo
- Tendo-se definido a partir de onde irá se contar, *deslocamento em bytes* determina de quantos bytes será o deslocamento a partir da posição atual.



Arquivo Binário – Exemplo 4 – fseek()

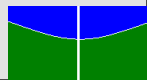
Exemplo: um programa que escreve um vetor de estruturas (struct) um arquivo com informações (código e nome) para 5 pessoas. Depois o programa acessa, lê, e mostra a terceira estrutura gravada no arquivo.

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

```
typedef struct {  
    int codigo;  
    char nome[30];  
} DadosDeFuncionario_t;
```

```
int main() {  
    FILE *fp;  
    DadosDeFuncionario_t dadosDeFuncionario[5], dadoslidos;  
    int i;  
    char nomeArquivo[50];  
  
    strcpy(nomeArquivo, "arqbin1.bin");  
    if (fopen(nomeArquivo, "a+b") == NULL) {  
        fp = fopen(nomeArquivo, "wb");  
    }  
}
```

```
/* CONTINUA NO PRÓXIMO SLIDE */
```



Arquivo Binário – Exemplo 4 – fseek()

/* CONTINUAÇÃO DO SLIDE ANTERIOR*/

```
for (i = 0; i < 5; i++ ) {  
    printf("Informe o nome: ");  
    scanf("%s", dadosDeFuncionario[i].nome);  
    printf("Informe o cod: ");  
    scanf("%d", &dadosDeFuncionario[i].codigo);  
}  
fwrite(&dadosDeFuncionario,sizeof(DadosDeFuncionario_t),5,fp);  
fclose(fp);  
fopen(nomeArquivo,"a+b");  
fseek (fp,(sizeof(DadosDeFuncionario_t)*2),0);  
fread(&dadoslidos, sizeof(DadosDeFuncionario_t),1,fp);  
printf("\n\nNome: %s",dadoslidos.nome);  
printf("\nCod: %d",dadoslidos.codigo);  
getchar();  
getchar();  
}
```


Grava no arquivo de uma vez
só todo o vetor com 5 estruturas.



Posiciona o ponteiro do arquivo
no início da 3a estrutura.



Lê um registro a partir da posição
onde o ponteiro se encontra,
no caso a 3a estrutura.



Como Mudar o Nome de um Arquivo

Para se mudar o nome de um arquivo usa-se a função ***rename()***:

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

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

O parâmetro *old* aponta para o nome de um arquivo existente cujo nome se deseja mudar para o nome apontado por *new*. Se um arquivo com nome *new* já existe, este será removido.

VALOR DE RETORNO

Caso a execução seja bem sucedida, *rename()* retornará 0; senão retornará -1 e a variável global *errno* conterá um valor inteiro correspondente ao erro, e nenhum dos arquivos *old* ou *new* será modificado ou criado.



Como Mudar o Nome de um Arquivo

Exemplo:

```
#include <errno.h>
```

```
int status;
```

```
char * nome1="/home/cnd/mod1";
```

```
char * nome2="/home/cnd/mod2";
```

```
status = rename(nome1, nome2);
```

```
if (status != 0)
```

```
    printf("Erro %d ao renomear o arquivo %s\n", errno, nome1);
```

```
else
```

```
    printf("Arquivo %d renomeado para %s\n", nome1, nome2);
```



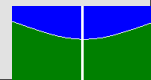
Trabalhando com Arquivos

- Para se encontrar a descrição da funcionalidade, valores de retorno e valores de erro de uma função em C procure no Google:

**man page <nome_da_função> ou
função <nome_da_função> em C**

- Um bom site para consultas rápidas é:

<http://www.mtm.ufsc.br/~azeredo/cursoC/c.html>



Trabalhando com Arquivos

- `int fgetc(FILE *fp)`
`char ch = fgetc(fp);`
- `int fputc(int c, FILE *fp)`
`fputc('a', fp);`

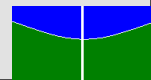


Passando Parâmetros na Linha de Comando

int main(int argc, char * argv[]) ou

int main(int argc, char ** argv)

- **argc** é um inteiro que indica quantos parâmetros foram passados na linha de comando (incluindo o nome do programa)
- **argv** é um vetor de strings que contém todas as strings passadas na linha de comando.



Passando Parâmetros na Linha de Comando

Exemplo 1: **\$ prog alfa 123 x 55a**

argc = 5

argv

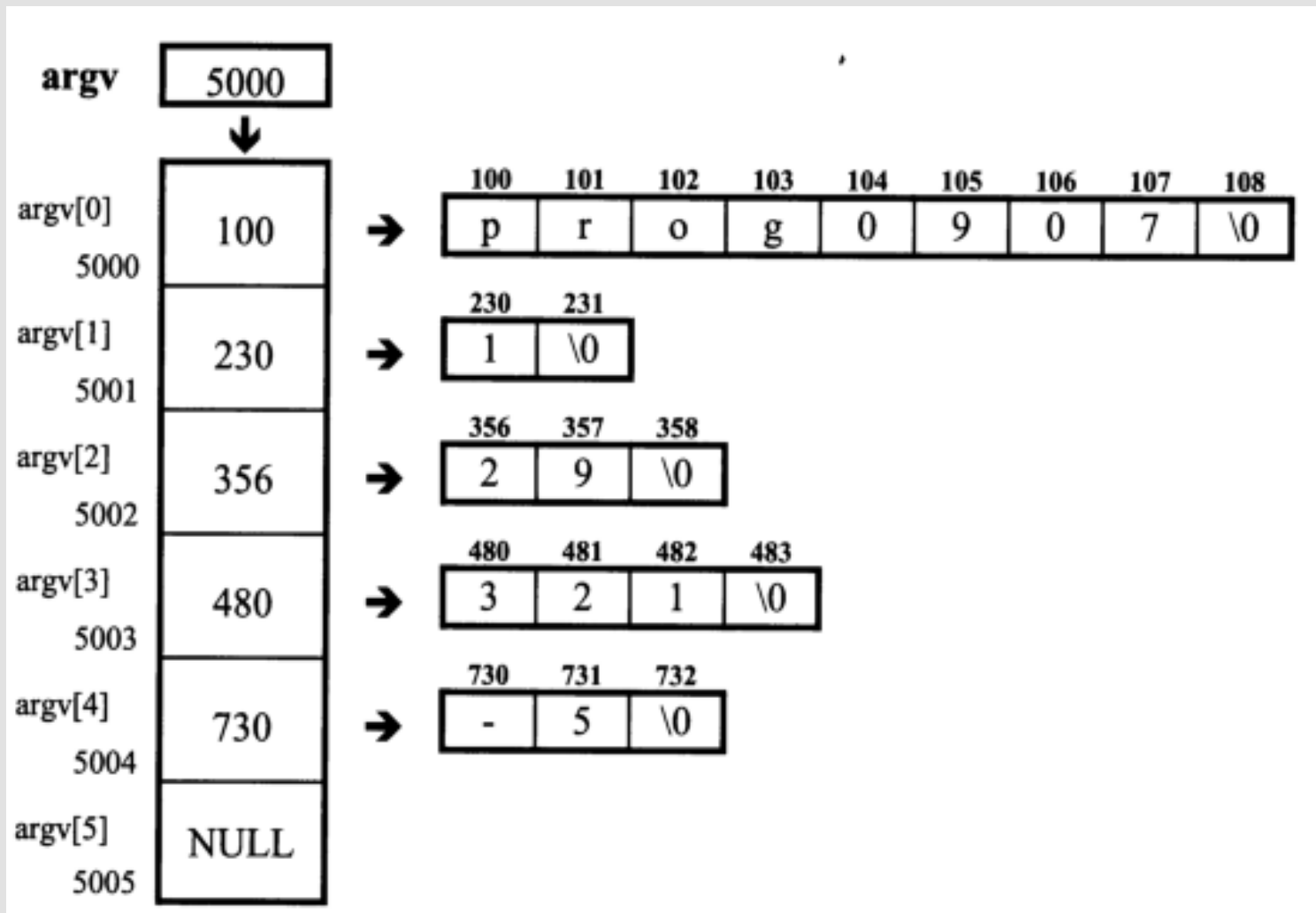
argv[0]	...	p	r	o	g	\0
argv[1]	...	a	l	f	a	\0
argv[2]	...	1	2	3	\0	
argv[3]	...	x	\0			
argv[4]	...	5	5	a	\0	
argv[5]	...	N	U	L	L	³

³A norma ANSI obriga que seja criada uma posição adicional no vetor argv onde se coloca o valor NULL.

Passando Parâmetros na Linha de Comando

Exemplo 2: **\$ prog097 1 29 321 -5**

argc = 5
argv



Passando Parâmetros na Linha de Comando

Problema 1: escreva um programa que mostre todos os parâmetros que recebeu na linha de comando.

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

int main (int argc, char * argv[])
{
    int i;
    system("cls");
    printf("\n\n");

    for (i=0; i < argc; i++) {
        printf("Parametro-%d: %s\n", i+1, argv[i]);
    }

    printf("\n\n");
    system("pause");
}
```

Passando Parâmetros na Linha de Comando

Problema 2: escreva um programa que some todos os números passados na linha de comando.

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

int main (int argc, char * argv[])
{
    int i, total=0;
    system("cls");
    printf("\n\n");

    for (i=1; i < argc; i++) {
        total += atoi(argv[i]);
    }

    printf("Total = %d\n\n", total);
    system("pause");
}
```

Passando Parâmetros na Linha de Comando

Problema 2b: escreva um programa que some todos os números passados na linha de comando. Use ponteiros.

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

int main (int argc, char * argv[])
{
    int i, total=0;
    system("cls");    printf("\n\n");
    argv++;

    for (; argv != NULL; argv++) {
        total += atoi(*argv);
    }

    printf("Total = %d\n\n", total);
    system("pause");
}
```