



LINGUAGEM C: ARQUIVOS (ADAPTADO DE PROF. ANDRÉ BACKES)

Prof. Martín Vigil

ARQUIVOS

- Por que usar arquivos?
 - Persistência dos dados;
 - Manipular grandes quantidade de informação;



TIPOS DE ARQUIVOS

- Basicamente, a linguagem C trabalha com dois tipos de arquivos: de **texto** e **binários**.
- Arquivo **binário**
 - armazena uma seqüência de **bits** que está sujeita as convenções dos programas que o gerou. Ex: arquivos executáveis, arquivos compactados, arquivos de registros, etc.
 - os dados são gravados na forma **binária** (do mesmo modo que estão na memória). Ex.: um número inteiro com 8 dígitos ocupará 32 bits no arquivo.



TIPOS DE ARQUIVOS

- Arquivo texto

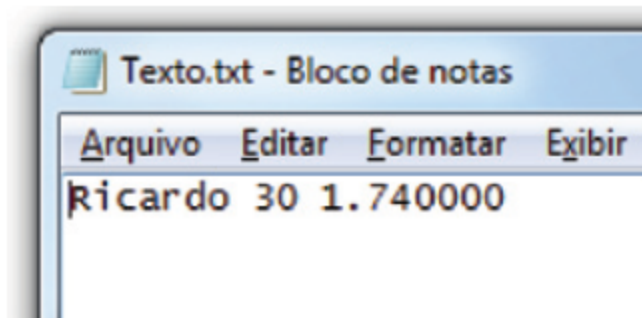
- armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de texto.
- Os dados são gravados como caracteres de 8 bits. Ex.: Um número inteiro de 8 dígitos ocupará 64 bits no arquivo (8 bits por dígito).



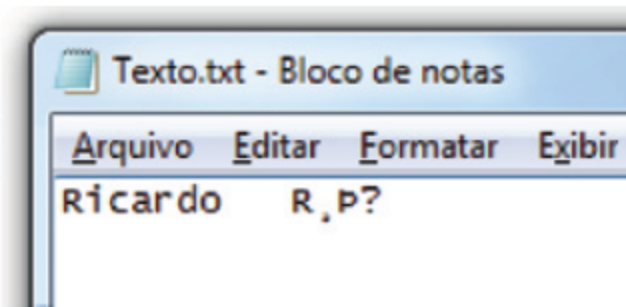
TIPOS DE ARQUIVOS

- Ex: Os dois trechos de arquivo abaixo possuem os mesmo dados :

```
char nome[20] = "Ricardo";  
int i = 30;  
float a = 1.74;
```



Arquivo Texto



Arquivo Binário



MANIPULANDO ARQUIVOS EM C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída, **stdio.h**.

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



MANIPULANDO ARQUIVOS EM C

- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo.
 - Suas funções se limitam a abrir/fechar e ler caracteres/bytes
 - É tarefa do programador criar a função que lerá um arquivo de uma maneira específica.



MANIPULANDO ARQUIVOS EM C

- Todas as funções de manipulação de arquivos trabalham com o conceito de "ponteiro de arquivo". Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

- **p** é o ponteiro para arquivo que nos permitirá manipular arquivos no C.



ABRINDO UM ARQUIVO

- Para a abertura de um arquivo, usa-se a função **fopen**

```
FILE *fopen(char *nome_arquivo, char *modo);
```

- O parâmetro **nome_arquivo** determina qual arquivo deverá ser aberto, sendo que o mesmo deve ser válido no sistema operacional que estiver sendo utilizado.



ABRINDO UM ARQUIVO

- No parâmetro **nome_arquivo** pode-se trabalhar com caminhos **absolutos** ou **relativos**.
 - **Caminho absoluto**: descrição de um caminho desde o diretório raiz.
 - C:\\Projetos\\dados.txt
 - **Caminho relativo**: descrição de um caminho desde o diretório corrente (onde o programa está salvo)
 - arq.txt
 - ../dados.txt

```
FILE *fopen(char *nome_arquivo, char *modo);
```



ABRINDO UM ARQUIVO

- O modo de abertura determina que tipo de uso será feito do arquivo.
- A tabela a seguir mostra os modo válidos de abertura de um arquivo.



ABRINDO UM ARQUIVO

Modo	Tipo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"rb"	Binário	Leitura. Arquivo deve existir.
"wb"	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"ab"	Binário	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+"	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+"	Texto	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+"	Texto	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+b"	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+b"	Binário	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+b"	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").

ABRINDO UM ARQUIVO

- Um arquivo binário pode ser aberto para escrita utilizando o seguinte conjunto de comandos:
 - A condição **fp==NULL** testa se o arquivo foi aberto com sucesso. No caso de erro a função **fopen()** retorna um ponteiro nulo (**NULL**).

```
int main() {  
    FILE *fp;  
    fp = fopen("exemplo.bin", "wb");  
    if(fp == NULL)  
        printf("Erro na abertura do arquivo.\n");  
  
    fclose(fp);  
  
    return 0;  
}
```



ERRO AO ABRIR UM ARQUIVO

- Caso o arquivo não tenha sido aberto com sucesso
 - Provavelmente o programa não poderá continuar a executar;
 - Nesse caso, utilizamos a função **exit()**, presente na biblioteca **stdlib.h**, para abortar o programa

```
void exit(int codigo_de_retorno);
```



ERRO AO ABRIR UM ARQUIVO

- A função **exit()** pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o **código_de_retorno**.
- A convenção mais usada é que um programa retorne
 - **zero** no caso de um término normal
 - um número **diferente de zero**, no caso de ter ocorrido um problema



ERRO AO ABRIR UM ARQUIVO

○ Exemplo

```
int main() {  
    FILE *fp;  
    fp = fopen("exemplo.bin", "wb");  
    if(fp == NULL) {  
        printf("Erro na abertura do arquivo\n");  
        system("pause");  
        exit(1);  
    }  
    fclose(fp);  
  
    return 0;  
}
```



POSIÇÃO DO ARQUIVO

- Ao se trabalhar com arquivos, existe uma espécie de posição onde estamos dentro do arquivo. É nessa posição onde será lido ou escrito o próximo caractere.
 - Quando utilizando o acesso seqüencial, raramente é necessário modificar essa posição.
 - Isso por que, quando lemos um caractere, a posição no arquivo é automaticamente atualizada.
 - Leitura e escrita em arquivos são parecidos com escrever em uma *máquina de escrever*



FECHANDO UM ARQUIVO

- Sempre que terminamos de usar um arquivo que abrimos, devemos fechá-lo. Para isso usa-se a função **fclose()**
- O ponteiro **fp** passado à função **fclose()** determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

```
int fclose(FILE *fp);
```



FECHANDO UM ARQUIVO

- Por que devemos fechar o arquivo?
 - Ao fechar um arquivo, todo caractere que tenha permanecido no "buffer" é gravado.
 - O "buffer" é uma região de memória que armazena temporariamente os caracteres a serem gravados em disco imediatamente. Apenas quando o "buffer" está cheio é que seu conteúdo é escrito no disco.



FECHANDO UM ARQUIVO

- Por que utilizar um “buffer”?? Eficiência!
 - Para ler e escrever arquivos no disco temos que posicionar a cabeça de gravação em um ponto específico do disco.
 - Se tivéssemos que fazer isso para cada caractere lido/escrito, a leitura/escrita de um arquivo seria uma operação muito lenta.
 - Assim a gravação só é realizada quando há um volume razoável de informações a serem gravadas ou quando o arquivo for fechado.
- A função **exit()** fecha todos os arquivos que um programa tiver aberto.



ESCRITA/LEITURA EM ARQUIVOS

- Uma vez aberto um arquivo, podemos ler ou escrever nele.
- Para tanto, a linguagem C conta com uma série de funções de leitura/escrita que variam de funcionalidade para atender as diversas aplicações.



ESCRITA/LEITURA DE CARACTERES

- A maneira mais fácil de se trabalhar com um arquivo é a leitura/escrita de um único caractere.
- A função mais básica de entrada de dados é a função **fputc** (*put character*).

```
int fputc (int ch, FILE *fp);
```


- Cada invocação dessa função grava um único caractere **ch** no arquivo especificado por **fp**.



ESCRITA/LEITURA DE CARACTERES

○ Exemplo da função **fputc**

```
int main() {  
    FILE *arq;  
    char string[100];  
    int i;  
    arq = fopen("arquivo.txt", "w");  
    if (arq == NULL) {  
        printf("Erro na abertura do arquivo");  
        system("pause");  
        exit(1);  
    }  
    printf("Entre com a string a ser gravada no arquivo:");  
    gets(string);  
    //Grava a string, caractere a caractere  
    for (i = 0; i < strlen(string); i++)  
        fputc(string[i], arq);  
    fclose(arq);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE CARACTERES

- A função **fputc** também pode ser utilizada para escrever um caractere na tela.
 - Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:
 - Por exemplo, **fputc ('*', stdout)** exibe um * na tela do monitor (dispositivo de saída padrão).

```
int main() {  
    fputc ('*', stdout);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE CARACTERES

- Da mesma maneira que gravamos um único caractere no arquivo, também podemos ler um único caractere.
- A função correspondente de leitura de caracteres é **fgetc** (*get character*).

```
int fgetc(FILE *fp);
```



ESCRITA/LEITURA DE CARACTERES

- Cada chamada da função **fgetc** lê um único caractere do arquivo especificado
 - Se **fp** aponta para um arquivo, então **fgetc(fp)** lê o caractere atual no arquivo e se posiciona para ler o próximo caractere do arquivo.
 - Lembre-se, a leitura em arquivos é parecida com escrever em uma *máquina de escrever*

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



ESCRITA/LEITURA DE CARACTERES

○ Exemplo da função **fgetc**

```
int main() {  
    FILE *arq;  
    char c;  
    arq = fopen("arquivo.txt", "r");  
    if (arq == NULL) {  
        printf("Erro na abertura do arquivo");  
        system("pause");  
        exit(1);  
    }  
    int i;  
    for (i = 0; i < 5; i++) {  
        c = fgetc(arq);  
        printf("%c", c);  
    }  
    fclose(arq);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE CARACTERES

- Similar ao que acontece com a função **fputc**, a função **fgetc** também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão):
- Nesse caso, **fgetc(stdin)** lê o próximo caractere digitado no teclado.

```
int main() {  
    char ch;  
    ch = fgetc(stdin);  
  
    printf("%c\n", ch);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE CARACTERES

- O que acontece quando **fgetc** tenta ler o próximo caractere de um arquivo que já acabou?
 - Precisamos que a função retorne algo indicando o arquivo acabou.
- Porém, todos os 256 caracteres são "válidos"!



ESCRITA/LEITURA DE CARACTERES

- Para evitar esse tipo de situação, **fgetc** não devolve um **char** mas um **int**:

```
int fgetc(FILE *fp);
```

- O conjunto de valores do **char** está contido dentro do conjunto do **int**.
 - Se o arquivo tiver acabado, **fgetc** devolve um valor **int** que não possa ser confundido com um **char**



ESCRITA/LEITURA DE CARACTERES

- Assim, se o arquivo não tiver mais caracteres, **fgetc** devolve o valor **-1**
- Mais exatamente, **fgetc** devolve a constante **EOF** (*end of file*), que está definida na biblioteca **stdio.h**. Em muitos computadores o valor de **EOF** é **-1**.

```
char c;  
c = fgetc(fp);  
if (c == EOF)  
    printf ("O arquivo terminou!\n");
```



ESCRITA/LEITURA DE CARACTERES

- Exemplo de uso do **EOF**

```
int main() {  
    FILE *arq;  
    char c;  
    arq = fopen("arquivo.txt", "r");  
    if (arq == NULL) {  
        printf("Erro na abertura do arquivo");  
        system("pause");  
        exit(1);  
    }  
    while ((c = fgetc(arq)) != EOF)  
        printf("%c", c);  
  
    fclose(arq);  
  
    return 0;  
}
```



ARQUIVOS PRÉ-DEFINIDOS

- Como visto anteriormente, os ponteiros **stdin** e **stdout** podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles **stdin** e **stdout**.



ARQUIVOS PRÉ-DEFINIDOS

- Alguns arquivos pré-definidos
 - **stdin**
 - dispositivo de entrada padrão (geralmente o teclado)
 - **stdout**
 - dispositivo de saída padrão (geralmente o vídeo)
 - **stderr**
 - dispositivo de saída de erro padrão (geralmente o vídeo)
 - **stdaux**
 - dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial)
 - **stdprn**
 - dispositivo de impressão padrão (em muitos sistemas, associado à porta paralela)



ESCRITA/LEITURA DE STRINGS

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo.
- Porém, existem funções na linguagem C que permitem ler/escrever uma seqüência de caracteres, isto é, uma string.
 - **fputs()**
 - **fgets()**



ESCRITA/LEITURA DE STRINGS

- Basicamente, para se escrever uma string em um arquivo usamos a função **fputs**:

```
int fputs(char *str, FILE *fp);
```

- Esta função recebe como parâmetro um array de caracteres (string) e um ponteiro para o arquivo no qual queremos escrever.



ESCRITA/LEITURA DE STRINGS

- Retorno da função
 - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.
 - Se houver erro na escrita, o valor EOF é retornado.
- Como a função **fputc**, **fputs** também pode ser utilizada para escrever uma string na tela:

```
int main() {  
    char texto[30] = "Hello World\n";  
    fputs(texto, stdout);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE STRINGS

○ Exemplo da função **fputs**:

```
int main() {  
    char str[20] = "Hello World!";  
    int result;  
    FILE *arq;  
    arq = fopen("ArqGrav.txt", "w");  
    if(arq == NULL) {  
        printf("Problemas na CRIACAO do arquivo\n");  
        system("pause");  
        exit(1);  
    }  
    result = fputs(str, arq);  
    if(result == EOF)  
        printf("Erro na Gravacao\n");  
    fclose(arq);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE STRINGS

- Da mesma maneira que gravamos uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo podemos usar a função **fgets()** cujo protótipo é:

```
char *fgets(char *str, int tamanho, FILE *fp);
```



ESCRITA/LEITURA DE STRINGS

- A função **fgets** recebe 3 parâmetros
 - **str**: aonde a lida será armazenada, **str**;
 - **tamanho** :o número máximo de caracteres a serem lidos;
 - **fp**: ponteiro que está associado ao arquivo de onde a string será lida.
- E retorna
 - NULL em caso de erro ou fim do arquivo;
 - O ponteiro para o primeiro caractere recuperado em **str**.

```
char *fgets(char *str, int tamanho, FILE *fp);
```



ESCRITA/LEITURA DE STRINGS

○ Funcionamento da função **fgets**

- A função lê a string até que um caractere de nova linha seja lido ou *tamanho-1* caracteres tenham sido lidos.
- Se o caractere de nova linha ('\n') for lido, ele fará parte da string, o que não acontecia com **gets**.
- A string resultante sempre terminará com '\0' (por isto somente *tamanho-1* caracteres, no máximo, serão lidos).
- Se ocorrer algum erro, a função devolverá um ponteiro nulo em **str**.



ESCRITA/LEITURA DE STRINGS


- A função **fgets** é semelhante à função **gets**, porém, com as seguintes vantagens:
 - Pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha “\n” na string;
 - Especifica o tamanho máximo da string de entrada. Isso evita estouro no buffer;



ESCRITA/LEITURA DE STRINGS

○ Exemplo da função **fgets**

```
int main() {  
    char str[20];  
    char *result;  
    FILE *arq;  
    arq = fopen("ArqGrav.txt", "r");  
    if (arq == NULL) {  
        printf("Problemas na ABERTURA do arquivo\n");  
        system("pause");  
        exit(1);  
    }  
    result = fgets(str, 13, arq);  
    if (result == NULL)  
        printf("Erro na leitura\n");  
    else  
        printf("%s", str);  
    fclose(arq);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE STRINGS

- Vale lembrar que o ponteiro **fp** pode ser substituído por **stdin**, para se fazer a leitura do teclado:

```
int main() {  
    char nome[30];  
    printf("Digite um nome: ");  
    fgets(nome, 30, stdin);  
    printf("O nome digitado foi: %s", nome);  
  
    return 0;  
}
```



ESCRITA/LEITURA DE BLOCO DE DADOS

- Além da leitura/escrita de caracteres e seqüências de caracteres, podemos ler/escrever blocos de dados.
- Para tanto, temos duas funções
 - **fwrite()**
 - **fread()**



ESCRITA/LEITURA DE BLOCO DE DADOS

- A função **fwrite** é responsável pela escrita de um bloco de dados da memória em um arquivo
- Seu protótipo é:

```
unsigned fwrite(void *buffer, int numero_de_bytes,  
               int count, FILE *fp);
```



ESCRITA/LEITURA DE BLOCO DE DADOS

- A função **fwrite** recebe 4 argumentos
 - **buffer:** ponteiro para a região de memória na qual estão os dados;
 - **numero_de_bytes:** tamanho de cada posição de memória a ser escrita;
 - **count:** total de unidades de memória que devem ser escritas;
 - **fp:** ponteiro associado ao arquivo onde os dados serão escritos.

```
unsigned fwrite(void *buffer, int numero_de_bytes,  
               int count, FILE *fp);
```



ESCRITA/LEITURA DE BLOCO DE DADOS

- Note que temos dois valores numéricos
 - **numero_de_bytes**
 - **count**
- Isto significa que o número total de bytes escritos é:
 - **numero_de_bytes * count**
- Como retorno, temos o número de unidades efetivamente escritas.
 - Este número pode ser menor que **count** quando ocorrer algum erro.



ESCRITA/LEITURA DE BLOCO DE DADOS: EXEMPLO FWRITE

```
5  int main(){
6      FILE *arq;
7      arq = fopen("ArqGrav1.txt","wb");
8
9      char str[20] = "Hello World";
10     float x = 5;
11     int v[5] = {1,2,3,4,5};
12
13     //grava str[0..strlen(str)]
14     fwrite(str,sizeof(char),strlen(str),arq);
15
16     //grava str[0..4]
17     fwrite(str,sizeof(char),5,arq);
18
19     //grava x
20     fwrite(&x,sizeof(float),1,arq);
21
22     //grava v[0..4]
23     fwrite(v,sizeof(int),5,arq);
24
25     //grava v[0..1]
26     fwrite(v,sizeof(int),2,arq);
27     fclose(arq);
28
29     return 0;
30 }
```



ESCRITA/LEITURA DE BLOCO DE DADOS: EXEMPLO FWRITE

str[0..11]="Hello world" str[0..4]="Hello"

ArqGrav1.txt

Offset	Hex Data	ASCII Data
0	48656C6C 6F20576F 726C6448 656C6C6F	Hello WorldHello
16	0000A040 01000000 02000000 03000000	†@
32	04000000 05000000 01000000 02000000	
48		

Signed Int little (select some data)

12 out of 48 bytes

x=5.0 v[0..4]={1,2,3,4,5} v[0..1]={1,2}

ESCRITA/LEITURA DE BLOCO DE DADOS

- A função **fread** é responsável pela leitura de um bloco de dados de um arquivo
- Seu protótipo é:

```
unsigned fread(void *buffer, int numero_de_bytes,  
               int count, FILE *fp);
```



ESCRITA/LEITURA DE BLOCO DE DADOS

- A função **fread** funciona como a sua companheira **fwrite**, porém lendo dados do arquivo.
- Como na função **fwrite**, **fread** retorna o número de itens escritos. Este valor será igual a **count** a menos que ocorra algum erro.

```
unsigned fread(void *buffer, int numero_de_bytes,  
               int count, FILE *fp);
```



ESCRITA/LEITURA DE BLOCO DE DADOS

○ Exemplo da função **fread**

```
char str1[20], str2[20];
float x;
int i, v1[5], v2[2];
//lê a string toda do arquivo
fread(str1, sizeof(char), 12, arq);
str1[12] = '\0';
printf("%s\n", str1);
//lê apenas os 5 primeiros caracteres da string
fread(str2, sizeof(char), 5, arq);
str2[5] = '\0';
printf("%s\n", str2);
//lê o valor de x do arquivo
fread(&x, sizeof(float), 1, arq);
printf("%f\n", x);
//lê todo o array do arquivo (5 posições)
fread(v1, sizeof(int), 5, arq);
for(i = 0; i < 5; i++)
    printf("v1[%d] = %d\n", i, v1[i]);
//lê apenas as 2 primeiras posições do array
fread(v2, sizeof(int), 2, arq);
for(i = 0; i < 2; i++)
    printf("v2[%d] = %d\n", i, v2[i]);
```



ESCRITA/LEITURA DE BLOCO DE DADOS

- Quando o arquivo for aberto para dados binários, **fwrite** e **fread** podem manipular qualquer tipo de dado.
 - int
 - float
 - double
 - array
 - struct
 - etc.



ESCRITA/LEITURA POR FLUXO PADRÃO

- As funções de fluxos padrão permitem ao programador ler e escrever em arquivos da maneira padrão com a qual o já líamos e escrevíamos na tela.
- As funções **fprintf** e **fscanf** funcionam de maneiras semelhantes a **printf** e **scanf**, respectivamente
- A diferença é que elas direcionam os dados para arquivos.



ESCRITA/LEITURA POR FLUXO PADRÃO

- Ex: **fprintf**

```
printf("Total = %d", x); //escreve na tela  
fprintf(fp, "Total = %d", x); //grava no arquivo fp
```

- Ex: **fscanf**

```
scanf("%d", &x); //lê do teclado  
fscanf(fp, "%d", &x); //lê do arquivo fp
```



ESCRITA/LEITURA POR FLUXO PADRÃO

○ Atenção

- Embora **fprintf** e **fscanf** sejam mais fáceis de ler/escrever dados em arquivos, nem sempre elas são as escolhas mais apropriadas.
- Como os dados são escritos em **ASCII e formatados** como apareceriam em tela, um tempo extra é perdido.
- Se a intenção é **velocidade**, deve-se utilizar as funções **fread** e **fwrite**.

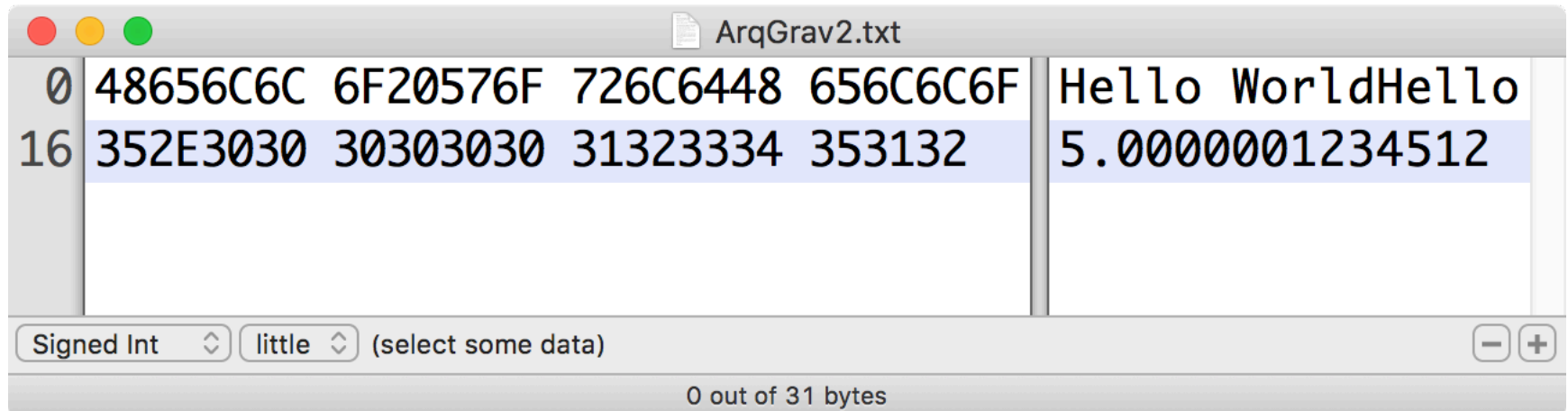


ESCRITA/LEITURA POR FLUXO PADRÃO, EXEMPLO FPRINTF

```
5 ▼ int main(){
6     FILE *arq;
7     arq = fopen("ArqGrav2.txt","wb");
8
9     char str[20] = "Hello World";
10    float x = 5;
11    int v[5] = {1,2,3,4,5};
12
13    //grava str[0..strlen(str)]
14    fprintf(arq, "%s", str);
15
16    //grava str[0..4]
17    fprintf(arq, "%.5s", str);
18
19    //grava x
20    fprintf(arq, "%f", x);
21
22    //grava v[0..4]
23    for(int i=0; i<5 ;i++ ) fprintf(arq, "%d", v[i]);
24
25    //grava v[0..1]
26    for(int i=0; i<2 ;i++ ) fprintf(arq, "%d", v[i]);
27
28    fclose(arq);
29
30    return 0;
31 }
```



ESCRITA/LEITURA POR FLUXO PADRÃO, RESULTADO FPRINTF VS. FWRITE



ESCRITA/LEITURA POR FLUXO PADRÃO, RESULTADO FPRINTF VS. FWRITE

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



ESCRITA/LEITURA POR FLUXO PADRÃO, RESULTADO FPRINTF VS. FWRITE

Resultado Fprintf

```
0  0048656C6C 006F20576F 007226C6448 006556C6C6F 0048656C6C 006F20576F 007226C6448 006556C6C6F
16 0048656C6C 006F20576F 007226C6448 006556C6C6F 0048656C6C 006F20576F 007226C6448 006556C6C6F

Signed Int little (select some data) 0 out of 31 bytes
```

```
0  0048656C6C 006F20576F 007226C6448 006556C6C6F 0048656C6C 006F20576F 007226C6448 006556C6C6F
16 0048656C6C 006F20576F 007226C6448 006556C6C6F 0048656C6C 006F20576F 007226C6448 006556C6C6F
32 0048656C6C 006F20576F 007226C6448 006556C6C6F 0048656C6C 006F20576F 007226C6448 006556C6C6F
48 0048656C6C 006F20576F 007226C6448 006556C6C6F 0048656C6C 006F20576F 007226C6448 006556C6C6F

Signed Int little (select some data) 12 out of 48 bytes
```


Resultado Fwrite

ESCRITA/LEITURA POR FLUXO PADRÃO

○ Exemplo da funções **fscanf**

```
int main() {
    FILE *arq;
    char texto[20], nome[20];
    int i;
    float a;
    int result;
    arq = fopen("ArqGrav.txt", "r");
    if(arq == NULL) {
        printf("Problemas na ABERTURA do arquivo\n");
        system("pause");
        exit(1);
    }
    fscanf(arq, "%s%s", texto, nome);
    printf("%s %s\n", texto, nome);
    fscanf(arq, "%s %d", texto, &i);
    printf("%s %d\n", texto, i);
    fscanf(arq, "%s%f", texto, &a);
    printf("%s %f\n", texto, a);
    fclose(arq);

    return 0;
}
```



MOVENDO-SE PELO ARQUIVO

- De modo geral, o acesso a um arquivo é seqüencial. Porém, é possível fazer buscas e acessos randômicos em arquivos.
- Para isso, existe a função **fseek**:

```
int fseek(FILE *fp, long numbytes, int origem);
```

- Basicamente, esta função move a posição corrente de leitura ou escrita no arquivo em tantos bytes, a partir de um ponto especificado.



MOVENDO-SE PELO ARQUIVO

- A função **fseek** recebe 3 parâmetros
 - **fp**: o ponteiro para o arquivo;
 - **numbytes**: é o total de bytes a partir de **origem** a ser pulado;
 - **origem**: determina a partir de onde os **numbytes** de movimentação serão contados.
- A função devolve o valor 0 quando bem sucedida

```
int fseek(FILE *fp, long numbytes, int origem);
```



MOVENDO-SE PELO ARQUIVO

- Os valores possíveis para **origem** são definidos por macros em **stdio.h** e são:

Nome	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto corrente do arquivo
SEEK_END	2	Fim do arquivo

- Portanto, para mover **numbytes** a partir
 - do início do arquivo, **origem** deve ser SEEK_SET
 - da posição atual, **origem** deve ser SEEK_CUR
 - do final do arquivo, **origem** deve ser SEEK_END
- numbytes** pode ser negativo quando usado com SEEK_CUR e SEEK_END



MOVENDO-SE PELO ARQUIVO

○ Exemplo da função **fseek**

```
struct cadastro{ char nome[20], rua[20]; int idade;};
int main(){
    FILE *f = fopen("arquivo.txt", "wb");

    struct cadastro c, cad[4] = {"Ricardo", "Rua 1", 31,
                                   "Carlos", "Rua 2", 28,
                                   "Ana", "Rua 3", 45,
                                   "Bianca", "Rua 4", 32};

    fwrite(cad, sizeof(struct cadastro), 4, f);
    fclose(f);

    f = fopen("arquivo.txt", "rb");
    fseek(f, 2*sizeof(struct cadastro), SEEK _ SET);
    fread(&c, sizeof(struct cadastro), 1, f);
    printf("%s\n%s\n%d\n", c.nome, c.rua, c.idade);
    fclose(f);

    return 0;
}
```



EXEMPLO2: BANCO DE DADOS

- Criar um arquivo para guardar registros de 10 empregados

32 bytes

id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario
id	nome	idade	salario

```
5  struct empregado {  
6      int id;  
7      char nome[20];  
8      int idade;  
9      float salario;  
10 };
```

Tamanho do arquivo = $32 * 10 = 320$ bytes



EXEMPLO2: BANCO DE DADOS

- Cria-se arquivo de 320 bytes contendo zeros
- Mostra-se conteúdo via hexdump

```
fwrite — -bash — 80x24
[machome:fwrite martin$ truncate -s 320 bancoDeDados.bin ]
[machome:fwrite martin$ hexdump -v -C bancoDeDados.bin ]
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000000a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000000b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000000c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000000d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000000e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000000f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000140
machome:fwrite martin$
```

EXEMPLO2: BANCO DE DADOS

- Funções para recuperar e gravar registro

```
12 struct empregado* getEmpregado(FILE *arq, int id){
13     fseek(arq, id*sizeof(struct empregado), SEEK_SET);
14     struct empregado *empregadoLido = (struct empregado *) calloc(1, sizeof(struct empregado));
15
16     if(fread(empregadoLido, sizeof(struct empregado), 1, arq) != 1)
17     {
18         printf("%s\n", "Erro de leitura");
19         exit(-1);
20     }
21
22     return empregadoLido;
23 }
24
25 void setEmpregado(FILE *arq, int id, struct empregado* umEmpregado)
26 {
27     fseek(arq, id*sizeof(struct empregado), SEEK_SET);
28     fwrite(umEmpregado, sizeof(struct empregado), 1, arq);
29 }
```

EXEMPLO2: BANCO DE DADOS

- Função para imprimir registro na tela

```
31 ▼ void printEmpregado(struct empregado* umEmpregado){  
32     printf("id: %d\n", umEmpregado->id);  
33     printf("nome: %s\n", umEmpregado->nome);  
34     printf("idade: %d\n", umEmpregado->idade);  
35     printf("salario: %f\n", umEmpregado->salario);  
36 }
```



EXEMPLO2: BANCO DE DADOS

- Abrindo o arquivo para escrita sem truncá-lo

```
38  int main(){  
39      FILE *arq;  
40      arq = fopen("bancoDeDados.bin","r+");
```



EXEMPLO2: BANCO DE DADOS

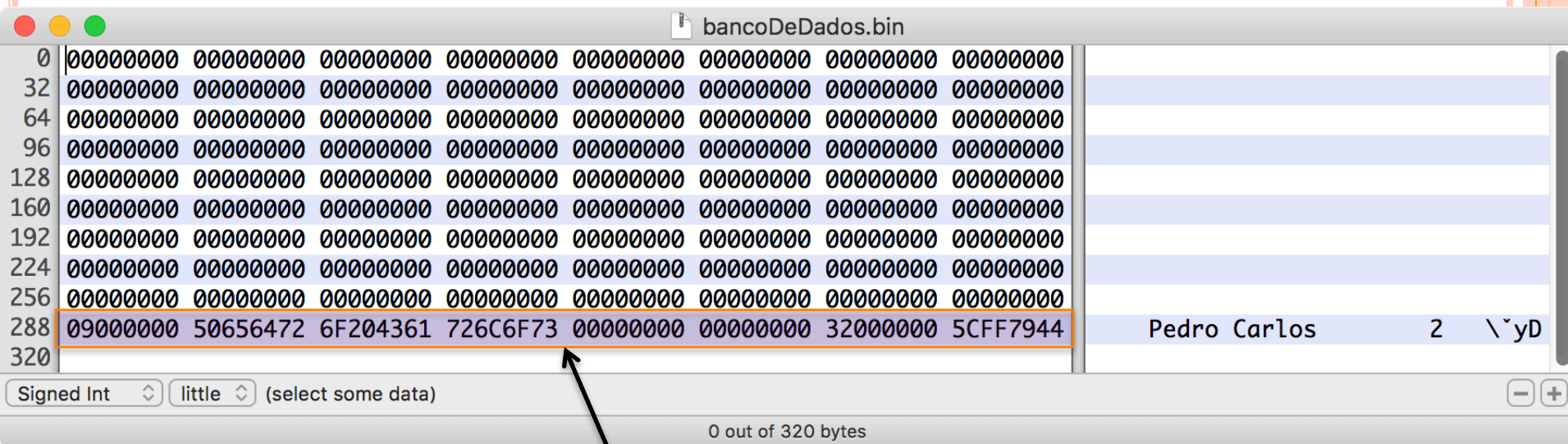
- Gravando empregado na última posição do arquivo (posição 9)

```
43 struct empregado *umEmpregado = (struct empregado*)calloc(1, sizeof(struct empregado));
44 umEmpregado->id=9;
45 strcpy(umEmpregado->nome, "Pedro Carlos");
46 umEmpregado->idade = 50;
47 umEmpregado->salario = 999.99;
48
49 setEmpregado(arq, umEmpregado->id, umEmpregado);
```



EXEMPLO2: BANCO DE DADOS

- Estado do arquivo após gravar registro



Registro inserido



EXEMPLO2: BANCO DE DADOS

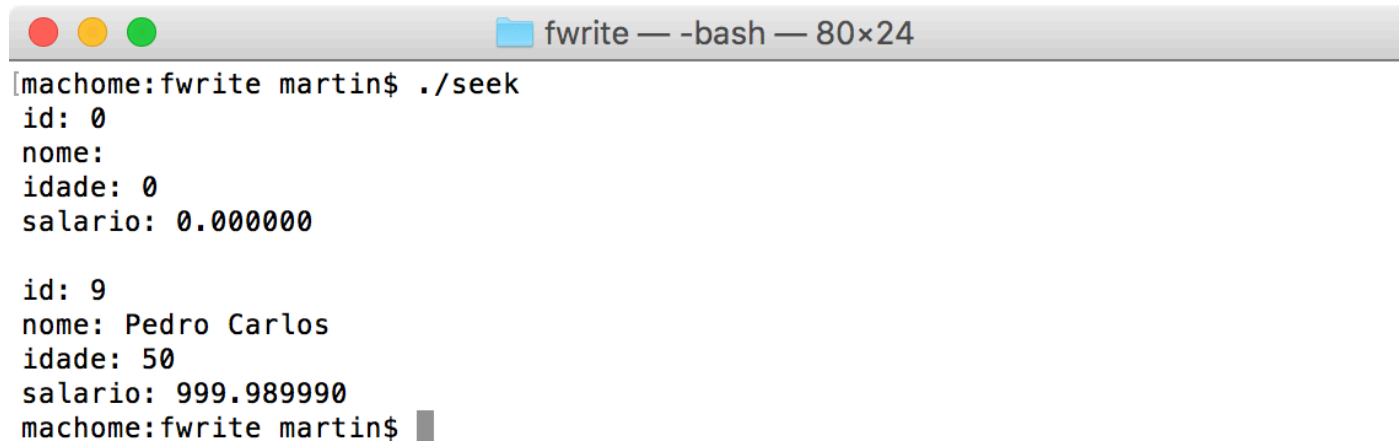
- Lendo e imprimindo empregados na posição 0 (vazio) e 9 (preenchida)

```
53      struct empregado *umEmpregado = getEmpregado(arq, 0);
54      printEmpregado(umEmpregado);
55      free(umEmpregado);
56
57      printf("\n");
58
59      umEmpregado = getEmpregado(arq, 9);
60      printEmpregado(umEmpregado);
61      free(umEmpregado);
```



EXEMPLO2: BANCO DE DADOS

- Lendo e imprimindo empregados na posição 0 (vazio) e 9 (preenchida)



```
fwrite — -bash — 80x24
[machome:fwrite martin$ ./seek
id: 0
nome:
idade: 0
salario: 0.000000

id: 9
nome: Pedro Carlos
idade: 50
salario: 999.989990
machome:fwrite martin$
```

MOVENDO-SE PELO ARQUIVO

- Outra opção de movimentação pelo arquivo é simplesmente retornar para o seu início.
- Para tanto, usa-se a função **rewind**:

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



DESCOBRINDO TAMANHO DE ARQUIVO

- Conhecer o tamanho do arquivo pode ser necessário. Ex: dimensionar buffers de leitura
- Como descobrir o tamanho do buffer?
 - Deslocar ponteiro até o final do arquivo via *seek*
 - Obter posição ponteiro via *ftell*



DESCRIPTION

The `fseek()` function sets the file position indicator for the stream pointed to by stream. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by whence. If whence is set to `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively. A successful call to the `fseek()` function clears the end-of-file indicator for the stream and undoes any effects of the `ungetc(3)` and `ungetwc(3)` functions on the same stream.

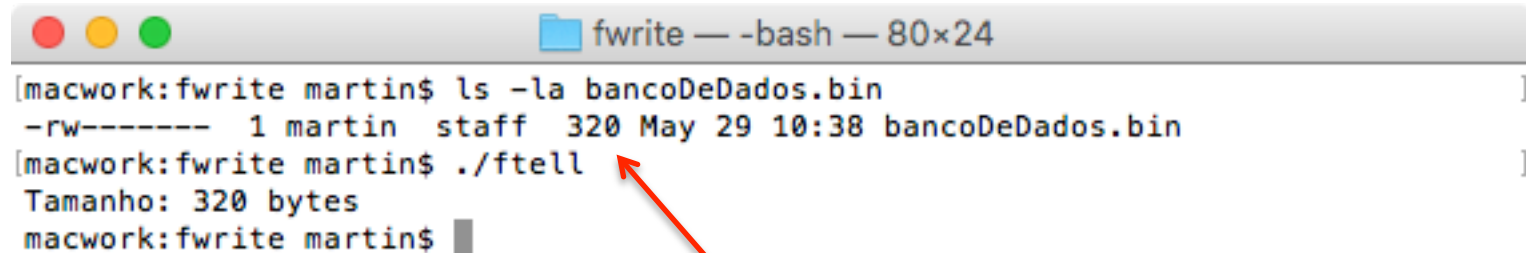
The `ftell()` function obtains the current value of the file position indicator for the stream pointed to by stream.



DESCOBRINDO TAMANHO DO ARQUIVO

```
1  #include <stdio.h>
2
3  long getSizeOfFile(FILE* arq){
4      fseek(arq, 0L, SEEK_END);
5      long size = ftell(arq);
6      rewind(arq);
7      return size;
8  }
9
10 int main(){
11     FILE *arq;
12     arq = fopen("bancoDeDados.bin","r");
13     printf("Tamanho: %ld bytes\n", getSizeOfFile(arq));
14 }
```

DESCOBRINDO TAMANHO DO ARQUIVO



A terminal window titled "fwrite — -bash — 80x24" with standard macOS window controls (red, yellow, green buttons). The terminal shows the following commands and output:

```
[macwork:fwrite martin$ ls -la bancoDeDados.bin ]  
-rw----- 1 martin  staff  320 May 29 10:38 bancoDeDados.bin ]  
[macwork:fwrite martin$ ./ftell ]  
Tamanho: 320 bytes  
macwork:fwrite martin$ █
```

A red arrow points from the "320" value in the output of the `ls` command to the "320 bytes" output of the `ftell` command.

APAGANDO UM ARQUIVO

- Além de permitir manipular arquivos, a linguagem C também permite apagá-lo do disco. Isso pode ser feito utilizando a função **remove**:

```
int remove(char *nome_do_arquivo);
```

- Diferente das funções vistas até aqui, esta função recebe o **caminho e nome** do arquivo a ser excluído, e não um ponteiro para FILE.
- Como retorno temos um valor inteiro, o qual será igual a 0 se o arquivo for excluído com sucesso.



APAGANDO UM ARQUIVO

- Exemplo da função **remove**

```
int main() {  
    int status;  
    status = remove("ArqGrav.txt");  
    if(status != 0) {  
        printf("Erro na remocao do arquivo.\n");  
        system("pause");  
        exit(1);  
    } else  
        printf("Arquivo removido com sucesso.\n");  
  
    return 0;  
}
```

