

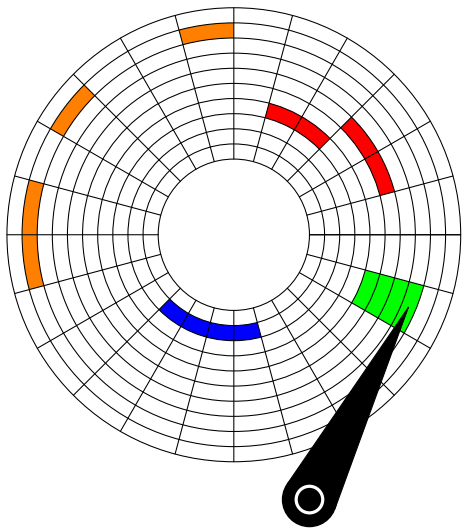
Arquivos em C

Agostinho Brito

2022

- 1 **Entrada e saída de dados**
- 2 **Aprimorando a leitura/escrita em arquivos - caracteres**
- 3 **Aprimorando a leitura/escrita em arquivos - literais**
- 4 **Aprimorando a leitura/escrita em arquivos - linhas**
- 5 **Lendo e escrevendo em modo binário**

Entrada e saída de dados



- Um arquivo é uma seção de armazenamento que recebe uma denominação específica.
- Os arquivos mais comuns são os arquivos de bloco, que ficam armazenados em unidades físicas: discos rígidos, unidades de estado sólido, fitas, CDs, DVDs, Blu-Rays...
- Arquivos são armazenados em sequências de blocos (fragmentos) de tamanho fixo agrupando setores.

O que são arquivos?

- Cada bloco pode conter informações adicionais para que o S.O. saiba como tratá-los.
- Parte do espaço em disco é usado para armazenar os dados úteis, parte é usada para preparar a infra-estrutura de indexação dos blocos.
- Num sistema Linux, os blocos possuem tamanho mínimo de 4096 bytes, e cada transferência disco ↔ memória totaliza 512 bytes, o tamanho de um setor.
- Agrupar setores (não necessariamente contíguos) em blocos diminui a demanda por endereços, permitindo discos maiores.

O que são arquivos?

- Cada bloco pode conter informações adicionais para que o S.O. saiba como tratá-los.
- Parte do espaço em disco é usado para armazenar os dados úteis, parte é usada para preparar a infra-estrutura de indexação dos blocos.
- Num sistema Linux, os blocos possuem tamanho mínimo de 4096 bytes, e cada transferência disco ↔ memória totaliza 512 bytes, o tamanho de um setor.
- Agrupar setores (não necessariamente contíguos) em blocos diminui a demanda por endereços, permitindo discos maiores.

Tá, mas e agora?

- Decidir o que fazer com com tantos blocos (que podem ser descontínuos) é algo além da alçada do programador de aplicativos comuns.
- Normalmente, desenvolvedores do kernel é quem cuidam dessa tarefa.
- É papel do sistema operacional facilitar para usuários e programadores o acesso às estruturas de um arquivo.
- Para isso, o S.O. oferece visões mais simplificadas de um arquivo regular no computador.

```
O rato roeu  
a roupa do  
rei de roma
```

O	_	r	a	t	o	_	r	o	e	u	↪	a	_	r	o	u	p	a	_	d	o	↪	r	e	i	_	d	e	_	r	o	m	a	EOF
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

- O arquivo nada mais é que uma sequência linear de caracteres, onde o final é apenas uma condição existente na estrutura do sistema de arquivos.

Modo binário e modo texto

- As bibliotecas do C padrão permitem que um arquivo seja visto de dois modos diferentes: binário e texto.
- Quando um arquivo é aberto no modo texto, algumas traduções são feitas para permitir interoperabilidade entre S.O.
- Por exemplo, no Linux, apenas o caractere `\n` (newline) é usado para marcar um fim de linha. No Windows, é usada a combinação `\r\n` (carriage return/newline).
- Quando o texto é aberto em modo **texto**, a sequência é lida apenas como `\n` (newline).
- Quando o texto é aberto em modo **binário**, a sequência é lida como `\r\n` (carriage return/newline).

Modo texto

Arquivos com texto pleno
(Códigos fonte, Texto
simples...)

Modo binário

Demais arquivos (Mídia,
Texto codificado, como
.doc ou .pdf)

- As bibliotecas do C padrão permitem que um arquivo seja visto de dois modos diferentes: binário e texto.
- Quando um arquivo é aberto no modo texto, algumas traduções são feitas para permitir interoperabilidade entre S.O.
- Por exemplo, no Linux, apenas o caractere `\n` (newline) é usado para marcar um fim de linha. No Windows, é usada a combinação `\r\n` (carriage return/newline).
- Quando o texto é aberto em modo **texto**, a sequência é lida apenas como `\n` (newline).
- Quando o texto é aberto em modo **binário**, a sequência é lida como `\r\n` (carriage return/newline).

Modo texto

Arquivos com texto pleno
(Códigos fonte, Texto
simples...)

Modo binário

Demais arquivos (Mídia,
Texto codificado, como
.doc ou .pdf)

- Em C, costuma-se utilizar duas funções bem comuns para ler dados do teclado e escrever dados numa tela: `scanf()` e `printf()`. Sua sintaxe é conhecida dos programadores desde o início do aprendizado da linguagem.
- Felizmente, o acesso aos arquivos também se dá com funções bem parecidas: `fscanf()` e `fprintf()`.
- Ambas precisam apenas que se forneça adicionalmente, um **PONTEIRO** para um `struct` que representará o arquivo a ser tratado.

```
FILE *f;  
fprintf(f, "3 + 4 = %d", 7);  
fscanf(f, "%d", &x);
```

- Entretanto, alguns procedimentos precisam ser feitos para que o arquivo seja corretamente manipulado...

Gravando o nome em um arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      if((file = fopen("meunome.txt", "w")) == NULL) {
6          exit(1);
7      }
8      fprintf(file, "Agostinho Brito\n");
9      fclose(file);
10 }
```

Gravando o nome em um arquivo

Para `exit()`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      if((file = fopen("meunome.txt", "w")) == NULL) {
6          exit(1);
7      }
8      fprintf(file, "Agostinho Brito\n");
9      fclose(file);
10 }
```

Gravando o nome em um arquivo

Para `exit()`

O ponteiro para `FILE`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      if((file = fopen("meunome.txt", "w")) == NULL) {
6          exit(1);
7      }
8      fprintf(file, "Agostinho Brito\n");
9      fclose(file);
10 }
```

Gravando o nome em um arquivo

Para `exit()`

O ponteiro para `FILE`

Abre o arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      if((file = fopen("meunome.txt", "w")) == NULL) {
6          exit(1);
7      }
8      fprintf(file, "Agostinho Brito\n");
9      fclose(file);
10 }
```

Gravando o nome em um arquivo

The diagram illustrates the process of writing a name to a file in C. It features a code block with four annotations in blue boxes with arrows pointing to specific lines of code:

- Para `exit()`**: Points to line 6, `exit(1);`.
- O ponteiro para FILE**: Points to line 4, `FILE *file;`.
- Abre o arquivo**: Points to line 5, `if((file = fopen("meunome.txt", "w")) == NULL) {`.
- Grava o nome no arquivo**: Points to line 8, `fprintf(file, "Agostinho Brito\n");`.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      if((file = fopen("meunome.txt", "w")) == NULL) {
6          exit(1);
7      }
8      fprintf(file, "Agostinho Brito\n");
9      fclose(file);
10 }
```

Gravando o nome em um arquivo

The diagram illustrates the process of writing a name to a file in C. It features a code block with line numbers 1 through 10. Five callout boxes with arrows point to specific parts of the code:

- Para `exit()`**: Points to `exit(1);` on line 6.
- O ponteiro para FILE**: Points to `FILE *file;` on line 4.
- Abre o arquivo**: Points to `fopen("meunome.txt", "w")` on line 5.
- Fecha o arquivo**: Points to `fclose(file);` on line 9.
- Grava o nome no arquivo**: Points to `fprintf(file, "Agostinho Brito\n");` on line 8.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      if((file = fopen("meunome.txt", "w")) == NULL) {
6          exit(1);
7      }
8      fprintf(file, "Agostinho Brito\n");
9      fclose(file);
10 }
```


Lendo o nome de um arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      char nome[50];
6      if((file = fopen("meunome.txt", "r")) == NULL) {
7          exit(1);
8      }
9      fscanf(file, "%s", nome);
10     printf("%s", nome);
11     fclose(file);
12 }
```

Lendo o nome de um arquivo

O ponteiro para FILE

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      char nome[50];
6      if((file = fopen("meunome.txt", "r")) == NULL) {
7          exit(1);
8      }
9      fscanf(file, "%s", nome);
10     printf("%s", nome);
11     fclose(file);
12 }
```

Lendo o nome de um arquivo

O ponteiro para FILE

Abre o arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      char nome[50];
6      if((file = fopen("meunome.txt", "r")) == NULL) {
7          exit(1);
8      }
9      fscanf(file, "%s", nome);
10     printf("%s", nome);
11     fclose(file);
12 }
```

Lendo o nome de um arquivo

O ponteiro para FILE

Abre o arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      char nome[50];
6      if((file = fopen("meunome.txt", "r")) == NULL) {
7          exit(1);
8      }
9      fscanf(file, "%s", nome);
10     printf("%s", nome);
11     fclose(file);
12 }
```

Lê o nome do arquivo

Lendo o nome de um arquivo

O ponteiro para FILE

Abre o arquivo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void){
4      FILE *file;
5      char nome[50];
6      if((file = fopen("meunome.txt", "r")) == NULL){
7          exit(1);
8      }
9      fscanf(file, "%s", nome);
10     printf("%s", nome);
11     fclose(file);
12 }
```

Fecha o arquivo

Lê o nome do arquivo

Modo	Significado
"r"	Abre o arquivo de texto para leitura.
"w"	Abre o arquivo de texto para escrita. Apaga o arquivo, caso exista, ou cria um novo se ele não existir.
"a"	Abre o arquivo para escrever após o final, ou cria um novo se ele não existir.
"r+"	Abre o arquivo para ler e escrever.
"w+"	Abre o arquivo para ler e escrever, apagando seu conteúdo caso ele exista.
"a+"	Abre o arquivo para ler e escrever, mas escrevendo só no final. O arquivo inteiro pode ser lido, mas a escrita só é permitida no fim.
"b"	Adicionando o caractere "b" a qualquer um dos modos, ativa o modo binário.

Modo	Significado
"r"	Abre o arquivo de texto para leitura.
"w"	Abre o arquivo de texto para escrita. Apaga o arquivo, caso exista, ou cria um novo se ele não existir.
"a"	Abre o arquivo para escrever após o final, ou cria um novo se ele não existir.
"r+"	Abre o arquivo para ler e escrever.
"w+"	Abre o arquivo para ler e escrever, apagando seu conteúdo caso ele exista.
"a+"	Abre o arquivo para ler e escrever, mas escrevendo só no final. O arquivo inteiro pode ser lido, mas a escrita só é permitida no fim.
"b"	Adicionando o caractere "b" a qualquer um dos modos, ativa o modo binário.



Praticando a manipulação de arquivos...



Obrigado

Aprimorando a leitura/escrita em arquivos - caracteres

- Para ler/escrever caracteres da/na entrada/saída padrão, C dispõe das funções `getchar()`/`putchar()`.
- Para ler/escrever caracteres de/em arquivos, C dispõe das funções `getc()`/`putc()`.

```
FILE *fin, *fout;  
c = getc(fin);  
putc(c, fout);
```

Fazendo a cópia de um arquivo...

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *fin, *fout;
5      int ch;
6      if((fin = fopen("entrada.txt", "r")) == NULL) {
7          exit(1);
8      }
9      if((fout = fopen("saida.txt", "w")) == NULL) {
10         exit(1);
11     }
12     while((ch = getc(fin)) != EOF) {
13         putc(ch, fout);
14     }
15     fclose(fout);
16 }
```



Obrigado

Aprimorando a leitura/escrita em arquivos - literais

- Para ler/escrever literais da/na entrada/saída padrão, C dispõe das funções `gets()`/`puts()`.



NUNCA USE `gets()`

- Para ler/escrever caracteres de/em arquivos, C dispõe das funções `fgets()`/`fputs()`.

```
FILE *fin, *fout;  
char buffer[50];  
fgets(buffer, 50, fin);  
fputs(buffer, fout);
```

- Se precisar ler da entrada padrão, use

```
fgets(fin, 50, stdin);
```



Obrigado

Aprimorando a leitura/escrita em arquivos - linhas

- E se quiséssemos ler uma linha inteira?
- A função `getline()` realiza essa tarefa, inclusive alocando o bloco de `char` necessário para armazenar a linha.

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);  
ssize_t getdelim(char **lineptr, size_t *n, int delim, FILE *stream);
```

- Se `lineptr` for igual a `NULL`, e `*n=0` antes da chamada, `getline()` aloca memória para armazenar o literal lido.
- Por outro lado, se `lineptr` for igual possui memória pré-alocada, `getline()` redimensiona a memória no tamanho necessário para armazenar a linha a ser lida.
- Em caso de redimensionamento, tanto `*lineptr` quanto `*n` serão atualizados.
- A função `getdelim()` funciona de modo semelhante, mas delimita o final da leitura pelo caractere fornecido em `delim`.
- Em caso de sucesso, retornam o número de caracteres lidos.
- Em caso de falha, retornam `-1`.
- A memória alocada **DEVE** ser liberada.

- E se quiséssemos ler uma linha inteira?
- A função `getline()` realiza essa tarefa, inclusive alocando o bloco de `char` necessário para armazenar a linha.

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);  
ssize_t getdelim(char **lineptr, size_t *n, int delim, FILE *stream);
```

- Se `lineptr` for igual a `NULL`, e `*n=0` antes da chamada, `getline()` aloca memória para armazenar o literal lido.
- Por outro lado, se `lineptr` for igual possui memória pré-alocada, `getline()` redimensiona a memória no tamanho necessário para armazenar a linha a ser lida.
- Em caso de redimensionamento, tanto `*lineptr` quanto `*n` serão atualizados.
- A função `getdelim()` funciona de modo semelhante, mas delimita o final da leitura pelo caractere fornecido em `delim`.
- Em caso de sucesso, retornam o número de caracteres lidos.
- Em caso de falha, retornam `-1`.
- A memória alocada **DEVE** ser liberada.

- E se quiséssemos ler uma linha inteira?
- A função `getline()` realiza essa tarefa, inclusive alocando o bloco de `char` necessário para armazenar a linha.

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);  
ssize_t getdelim(char **lineptr, size_t *n, int delim, FILE *stream);
```

- Se `lineptr` for igual a `NULL`, e `*n=0` antes da chamada, `getline()` aloca memória para armazenar o literal lido.
- Por outro lado, se `lineptr` for igual possui memória pré-alocada, `getline()` redimensiona a memória no tamanho necessário para armazenar a linha a ser lida.
- Em caso de redimensionamento, tanto `*lineptr` quanto `*n` serão atualizados.
- A função `getdelim()` funciona de modo semelhante, mas delimita o final da leitura pelo caractere fornecido em `delim`.
- Em caso de sucesso, retornam o número de caracteres lidos.
- Em caso de falha, retornam `-1`.
- A memória alocada **DEVE** ser liberada.

- E se quiséssemos ler uma linha inteira?
- A função `getline()` realiza essa tarefa, inclusive alocando o bloco de `char` necessário para armazenar a linha.

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);  
ssize_t getdelim(char **lineptr, size_t *n, int delim, FILE *stream);
```

- Se `lineptr` for igual a `NULL`, e `*n=0` antes da chamada, `getline()` aloca memória para armazenar o literal lido.
- Por outro lado, se `lineptr` for igual possui memória pré-alocada, `getline()` redimensiona a memória no tamanho necessário para armazenar a linha a ser lida.
- Em caso de redimensionamento, tanto `*lineptr` quanto `*n` serão atualizados.
- A função `getdelim()` funciona de modo semelhante, mas delimita o final da leitura pelo caractere fornecido em `delim`.
- Em caso de sucesso, retornam o número de caracteres lidos.
- Em caso de falha, retornam `-1`.
- A memória alocada **DEVE** ser liberada.

Lendo um arquivo linha a linha

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(void) {
4      FILE *file;
5      char *linha = NULL;
6      size_t tam = 0;
7      int nlidos;
8      if((file = fopen("entrada.txt", "r")) == NULL) {
9          exit(1);
10     }
11     while ((nlidos = getline(&linha, &tam, file)) != -1) {
12         printf("%s", linha);
13     }
14     free(linha);
15     fclose(file);
16 }
```



Obrigado

Lendo e escrevendo em modo binário

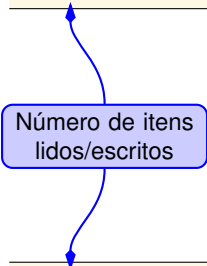
- Escrita em modo texto é conveniente, visto que é sempre possível visualizar o que se escreveu.
- Entretanto, para alguns tipos de aplicações, o modo texto gera dados adicionais que podem ser desnecessários.
- Ex: O número 236, se escrito em um arquivo ocupará 3 bytes, mas se o seu formato for `unsigned char`, apenas 1 byte apenas é necessário para representá-lo.
- A solução para isso é usar processod de escrita em modo binário.
- Nesse caso, o par de funções `fread()` e `fwrite()` pode ser usada para operações de Entrada/Saída.


```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Número de itens
lidos/escritos



```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Número de itens
lidos/escritos

Ponteiro para o
início dos dados

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Número de itens
lidos/escritos

Ponteiro para o
início dos dados

Bytes
por item

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Lendo e escrevendo em modo binário

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Número de itens
lidos/escritos

Ponteiro para o
início dos dados

Bytes
por item

Número de itens

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Lendo e escrevendo em modo binário

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Número de itens
lidos/escritos

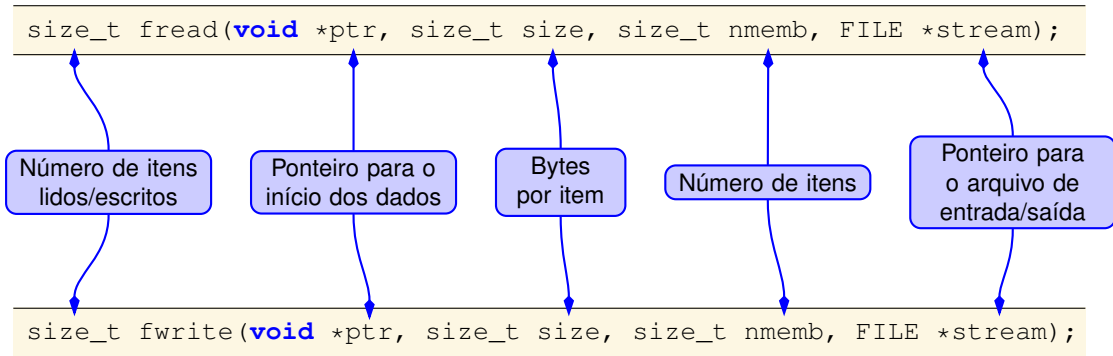
Ponteiro para o
início dos dados

Bytes
por item

Número de itens

Ponteiro para
o arquivo de
entrada/saída

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```



Praticando fread()/fwrite()...



Obrigado