Funções da biblioteca `stdio.h` a serem exercitadas hoje:

```
FILE *fopen(const char *caminho, const char *modo);
```

A função `fopen` abre o arquivo cujo nome é a *string* `caminho` e retorna a *stream* associada a ele ou `NULL` em caso de erro.

O argumento `modo` é uma string que começa com uma ou mais sequências abaixo:

- **`r`** Abre arquivo para leitura. O indicador de posição é colocado no início do arquivo.
- **`r+`** Abre arquivo para leitura e escrita. O indicador de posição é colocado no início do arquivo.
- **`w`** Cria o arquivo (se não existe) ou trunca-o para comprimento zero (se já existe) e abre-o apenas para escrita. O indicador é colocado no início do arquivo.
- **`w+`** Cria ou trunca o arquivo e abre-o para leitura e escrita. O indicador é colocado no início do arquivo.
- **`a`** Abre para escrita, sem truncar. Se o arquivo não existe, é criado. Indicador é colocado no final do arquivo.
- **`a+`** Abre para leitura e escrita, sem truncar. Se o arquivo não existe, é criado. Indicador é colocado no final do arquivo.

Por padrão, o arquivo é aberto em **modo texto**, o que pode fazer com que algumas sequências de bytes possam ser tratadas de maneira especial (comumente os caracteres de fim de linha).

Opcionalmente, a letra "**`b`**" pode aparecer como último caractere na *string* `modo` para indicar que o arquivo deve ser aberto em **modo binário**.

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

Fecha um arquivo aberto com `fopen`. Retorna 0 caso a operação seja bem sucedida ou `EOF` em caso de erro.

```
int fprintf(FILE *stream, const char *formato, ...);
```

```
int fscanf(FILE *stream, const char *formato, ...);
```

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

Versões de `printf`, `scanf` e `puts` que escrevem na/leem da *stream* passada como parâmetro.

```
char *fgets(char *s, int tam, FILE *stream);
```

Lê no máximo `tam`-1 bytes da stream e armazena-os na string `s`. A leitura termina ao encontrar quebra de linha ("\n") ou fim de arquivo. Se a quebra de linha for lida, ela é colocada na string. O caractere nulo ("\0") é colocado na string após o último byte lido. A função retorna um ponteiro para a string caso pelo menos 1 byte seja lido, ou `NULL` caso nenhum byte seja lido antes do fim do arquivo ser encontrado.

```
size_t fread(void *ptr, size_t tam, size_t nelt, FILE *stream);
```

Lê `nelt` elementos de dados, cada um com `tam` bytes, armazenando-os no local apontado por `ptr`. Retorna o número de itens lidos, que pode ser `nelt` se todos os elementos requisitados foram lidos, ou um número menor que `nelt` caso ocorra um erro ou fim de arquivo antes que os `nelt` elementos possam ser lidos.

```
size_t fwrite(void *ptr, size_t tam, size_t nelt, FILE *stream);
```

Escreve `nelt` elementos de dados, cada um com `tam` bytes, obtendo-os do local apontado por `ptr`. Retorna o número de itens escritos, que pode ser `nelt` se todos os elementos fornecidos foram escritos, ou um número menor que `nelt` caso ocorra um erro antes que os `nelt` elementos possam ser escritos.

`int fseek(FILE *stream, long desloc, int apartirde);`

Ajusta o indicador de posição do arquivo. A nova posição, medida em bytes, é obtida pelo acréscimo de `desloc` bytes à posição especificada por `apartirde`, que pode ser uma das seguintes constantes:

- `SEEK_SET`, deslocamento relativo ao início do arquivo
- `SEEK_CUR`, deslocamento relativo à posição atual
- `SEEK_END`, deslocamento relativo à posição do fim do arquivo

A função `fseek` retorna 0 se a operação é bem sucedida ou -1 em caso de erro (por exemplo, se o deslocamento leva o indicador de posição para antes do início do arquivo ou para após o final do arquivo).

`long ftell(FILE *stream);`

Retorna o valor atual do indicador de posição, em bytes a partir do início do arquivo.

Exercício 1

Complete o programa `invertelinhas.c`. Ele receberá dois parâmetros de linha de comando: um arquivo de entrada e um arquivo de saída. O programa deve ler o arquivo de entrada, linha por linha, e gravar no arquivo de saída cada uma das linhas lidas com os caracteres em ordem reversa.

Considere que nenhuma linha possui mais de 80 bytes, incluindo os caracteres de quebra de linha.

Para testar, pegue o arquivo `ykcowrebbaj.txt`. Exemplo de execução:

```
$ ./invertelinhas ykcowrebbaj.txt invertido.txt
```

O arquivo `invertido.txt` terá o mesmo conteúdo que o arquivo `jabberwocky.txt` fornecido.

Exercício 2

Considerando a struct definida em `produto.h`, complete o programa `converteprodutos.c` para que ele leia uma lista de produtos no formato de valores separados por tabulações com id, nome, unidade, marca e preço (exemplo em `lista.csv`) e grave o conteúdo de cada struct lida diretamente em um arquivo binário.

Exemplo de execução:
```
$ ./converteprodutos lista.csv saida.bin
```

Exercício 3

Considerando a struct definida em `produto.h`, complete o programa `listaproduto.c` para que ele abra um arquivo binário como o gerado no exercício anterior e imprima o i-ésimo elemento da lista. O acesso ao elemento deve ser direto, sem que os elementos anteriores sejam lidos.

Exemplo de execução:

```
$ ./listaproduto saida.bin 3
ID: 3
Produto: Detergente
Unidade: 500ml
Marca: Ypê Clear
Preço: R$ 1.75
```

Note que os identificadores de produto são inteiros em sequência começando em 1.

Exercício 4 (para casa)

Faça um programa para cadastrar um novo produto no arquivo binário e gravá-lo no final do arquivo com o próximo ID disponível.

Exemplo de execução: (o texto em negrito é digitado pelo usuário)

```
$ ./cadastraproduto
Produto: Pipoca para microondas
Unidade: 100g
Marca: Yoki
Preço: R$ 2.35
Produto cadastrado com ID 40
```