Questão 2) Contexto: Organização de Biblioteca Digital
Uma biblioteca digital está implementando um sistema de controle de acesso e organização dos seus livros e leituras. O sistema deve lidar com as seguintes operações:

I. **Solicitações de Empréstimo:** Leitores fazem pedidos para empréstimo de livros. Os pedidos são processados na ordem em que chegam.

II. **Histórico de Leituras Recentes:** Ao navegar por livros, o sistema mantém um histórico dos últimos acessos para permitir que o usuário volte às leituras anteriores, de forma reversa.

III. **Catálogo Geral de Livros:** O acervo completo da biblioteca deve permitir acesso, remoção e inserção de livros em qualquer posição, possibilitando a organização alfabética ou por categorias, priorizando o acesso. Inserções e remoções são operações pouco frequentes.

Determine qual a estrutura de dados e forma de implementação **mais adequada para as operaçoes abaixo.** Justifique suas escolhas e defina -as.

I. Solicitações de Empréstimo

Estrutura de dados mais adequada: Fila (Queue).

- **Forma de implementação:** encadeada, já que inserções (no fim) e remoções (no início) são frequentes.
- **Justificativa:**
  As solicitações precisam ser atendidas **na ordem em que chegam** (regra FIFO — First In, First Out). A fila organiza as reservas conforme a ordem de chegada, garantindo que o primeiro a solicitar seja o primeiro a ser atendido.

```
typedef struct no {
    Emprestimo * dado;
    struct no *prox;
} No;

typedef struct {
    No *inicio;
    No *fim;
    int tamanho;
} Fila;
```

II. Histórico de Leituras

Estrutura de dados mais adequada: Pilha (Queue).

- **Forma de implementação:** encadeada, já que inserções e remoções são frequentes.
- **Justificativa:**
  O histórico precisa registrar o equipamento ou livro mais recente primeiro. Assim, o acesso ao último elemento adicionado deve ser rápido (LIFO — Last In, First Out), o que a pilha permite. Inserções e remoções ocorrem sempre no topo.

```
typedef struct {
    Empréstimo *topo;
    int tamanho;
} Pilha;
```

III. Catálogo Geral de Livros

Estrutura de dados mais adequada: Lista sequencial classificada

- **Forma de implementação: sequencial** (vetor) , ordenado alfabeticamente pelos nomes dos livros.
- **Justificativa:**
  O acesso é frequente e rápido. Como inserções e remoções são menos frequentes, o custo de realocar elementos é aceitável. Ordenar alfabeticamente facilita buscas sequenciais binárias

```
typedef struct livro {
    char titulo[100];    // nome do livro
    char autor[50];      // nome do autor
    int ano;             // ano de publicação
} Livro;

typedef struct {
    Livro*livros    ;  // vetor para os livros
    int max;          // quantidade máxima de livros cadastrados
    int qtde;         // quantidade atual de livros cadastrados
} Catalogo;
```

Considere que o TAD MATRIZ, definido abaixo, está disponivel no arquivo Tad_Matriz.h

O TAD Matriz (Tipo Abstrato de Dados para Matrizes) representa uma estrutura de dados bidimensional composta por elementos dispostos em linhas e colunas, geralmente acessados por dois índices: linha e coluna.

Operações do TAD Matriz:

- *criar_matriz(linhas, colunas)*: retorna o endereço de uma matriz dinamicamente alocada com as dimensões fornecidas.
- *set_valor(matriz*, i, j, valor):* insere ou atualiza o valor na posição [i][j].
- *get_valor(matriz*, i, j)*: retorna o valor da posição [i][j].
- *soma(matriz1*, matriz2)*: retorna o endereço de uma nova matriz resultante da soma elemento a elemento.
- *multiplicar(matriz1*, matriz2*)*: retorna o produto matricial.
- *transposta(matriz*)*: devolve a matriz transposta.
- *exibir_matriz(matriz*)*: exibe todos os elementos da matriz em formato tabular, com linhas e colunas numeradas.
- *clona_linha(matriz*, linha)*: devolve uma nova matriz com 1 linha e n colunas, preenchidos com os valores das células da linha especificada
- *clona_coluna(matriz*, coluna)*: devolve uma nova matriz com n linhas e 1 coluna, preenchidos com os valores das células da coluna especificada
- *soma_linha(matriz*, linha)*: devolve a soma dos valores das células da linha especificada
- *soma_coluna(matriz*, coluna)*: devolve a soma dos valores das células da coluna especificada
- *liberar_matriz(matriz*)*: desaloca a memória

Usando o TAD Matriz acima e a função preenche_acessos (cujo cabeçalho você deve definir, mas não precisa implementá-la) que preenche a estrutura recebida pela função com o número de acessos à página por dia x hora **resolva o seguinte problema:**

Um site deseja monitorar o número de acessos em sua página ao longo da semana. Para isso, armazena os dados em uma matriz de inteiros 7 x 24, onde:

- Cada linha representa um dia da semana (0 a 6 → de domingo a sábado),
- Cada coluna representa uma hora do dia (0 a 23)

Construa um programa, usando a preenche_acessos e o TAD Matriz, que calcule e mostre:

- a) O total de acessos em um determinado dia;
- b) A média de acessos por hora ao longo da semana; (média das 0h, 1h,2h,...)
- c) Um dia e a hora com o maior número de acessos.

```
#include "Tad_Matriz.h"

#define DIAS 7
#define HORAS 24

// Cabeçalho da função (a implementação será fornecida externamente)
void preenche_acessos(Matriz *m);

int main() {

    Matriz *acessos = criar_matriz(DIAS, HORAS);
    preenche_acessos(acessos);
    printf("Matriz de acessos (Dia x Hora):\n");
    exibir_matriz(acessos);

    // a) Total de acessos em um dia informado
    int dia;
```

```
    printf("Informe o dia da semana (0 a 6): ");
    scanf("%d", &dia);
    int total = soma_linha(acessos, dia);
    printf("Total de acessos no dia %d: %d\n", dia, total);

    // b) Média de acessos por hora ao longo da semana (por dia)
    printf("\nMédia de acessos por hora em cada dia:\n");
    for (int i = 0; i < HORAS; i++) {
        int soma = soma_coluna(acessos, i);
        float media = soma / (float)DIAS;
        printf("Dia %d: Média = %.2f acessos/dia\n", i, media);
    }

    // c) Dia e hora com o maior número de acessos
    int max = -1, dia_max = 0, hora_max = 0;
    for (int i = 0; i < DIAS; i++) {
        for (int j = 0; j < HORAS; j++) {
            int val = get_valor(acessos, i, j);
            if (val > max) {
                max = val;
                dia_max = i;
                hora_max = j;
            }
        }
    }
    printf("\nMaior número de acessos: %d no dia %d às %02dh\n", max, dia_max,
hora_max);

    liberar_matriz(acessos);
    return 0;
}
```

Considere que foi implementado um TAD para uma **lista simplesmente encadeada**, **sem nó cabeça**, contendo inteiros ordenados em **ordem crescente**:

Estrutura do nó:

```
struct No {
    int valor;
    struct No *prox;
} ;
typedef struct No tNo;
```

Estrutura da lista:

```
struct l {
    int qt;
    tNo  *prim;
} ;
typedef struct L tlista;
```

Construa a função ***inverter_lista*** que, sem criar novos nós e sem realocar os valores dos campos valor, apenas alterando os ponteiros (prox), **reorganize a lista** para que os nós passem a estar em **ordem decrescente**.

Protótipo: `void inverter_lista(tLista* l);`

Exemplo: Lista original (ordem crescente): [1] → [3] → [5] → [7] → NULL

Após a chamada:Lista modificada (ordem decrescente): [7] → [5] → [3] → [1] → NULL

```
inverter_lista(tLista* l) {
    No* anterior = NULL;
    No* atual = l->prim;
    No* proximo;

    while (atual != NULL) {
        proximo = atual->prox;   // guarda o próximo nó
        atual->prox = anterior;  // inverte o ponteiro do nó atual
        anterior = atual;        // move o anterior para o atual
        atual = proximo;         // avança na lista
    }
    l->prim= anterior
    return
}
```