	CENTRO UNIVERSITÁRIO DE JOÃO PESSOA PRÓ-REITORIA ACADÊMICA - PROAC CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO			
	Componente Curricular: ESTRUTURA DE DADOS I			
	Professor: WALACE SARTORI BONFIM			
	Semestre: 2023.1	Período: 3º	Turma:ED I P2B	Data:

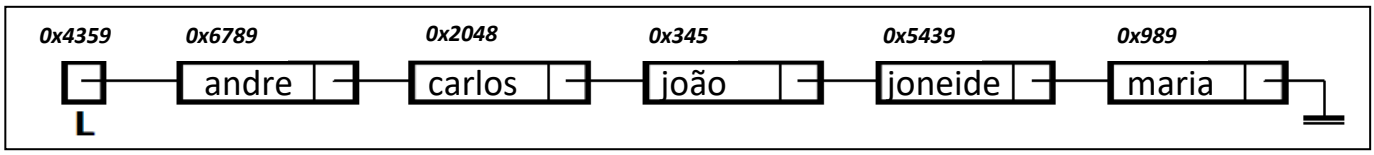
Aluno: Marcelo Camilo Gomes
RGM: 32004061

EXERCÍCIO DE FIXAÇÃO PARA A 1ª AVALIAÇÃO

EXERCÍCIO 01

Com objetivo de distinguir conceitos de ponteiros e utilizar corretamente os operadores "&", "->", "*" e ".", considere a figura que apresenta o contexto de uma Agenda Eletrônica implementada utilizando uma **1. Lista Encadeada com 5 nós** e as respectivas **2.Declarações das estruturas, tipos e variáveis**. Você deve ler atentamente cada afirmativa feita na seção **3. Dez afirmativas** e responder V ou F para cada **4. Combinações de afirmativas**.

1. LISTA ENCADEADA COM 5 NÓS



2. DECLARAÇÕES DAS ESTRUTUAS, TIPOS E VARIÁVEIS:

```
// Tipo base dos elementos da lista
typedef struct elementos {
    char nome[50];
} t_elemento;

// Estrutura da lista
typedef struct no {
    t_elemento dado;  // elemento contendo os dados
    struct no  * prox; // ponteiro para o proximo elemento
} t_no; // tipo da estrutura

typedef t_no * t_lista; // t_lista é igual a t_no *

t_lista L, M;
```

3. DEZ AFIRMATIVAS

1) L é uma variável que carrega dentro dela um endereço de memória de uma área do tipo t_lista; F - L contém um endereço de memória do tipo t_no

2) A atribuição M = L; resulta em colocar o valor 0x4359 dentro da variável M; F - seria 0x6789

3) O operador "&" aplicado em L, &L, resulta em 0x4359; V

4) O operador "*" aplicado em L, *L, resulta em uma área do tipo t_no alocada na posição 0x6789 da memória; V

5) O operador "->" aplicado em L, L->prox, resulta em 0x5439 e L->prox->prox resulta em 0x6789; F - mt errado

6) O operador “->” aplicado em L, L->dado, resulta em uma área do tipo t_elemento que carrega dentro dela um vetor de 50 caracteres; V

7) Os operadores “->” e “.” aplicados em L e na estrutura contida em L, (L->prox)->dado.nome, resulta no string “carlos”; V

8) Para percorrer a lista acessando cada nó, basta utilizar uma variável do tipo t_lista, seja Pto, e realizar a operação Pto = L->prox; F - Pto = Pto->prox em loop

9) A função FUNC(t_lista P1, t_lista * P2, t_elemento DD) que for chamada como FUNC(L, &L, L->prox.dado) irá receber em P1 o valor 0x6789, em P2 o valor 0x4359 e em DD a área do tipo t_elemento que está na posição 0x6789 de memória; V

10) Para partir de L e chegar na área de endereço 0x2048 tenho que inicializar Pto com o valor de L, seja Pto = L, fazer um loop de três repetições e dentro do laço colocar a instrução Pto = Pto->prox; F - duas repetições

1F
2F
3V
4V
5F
6V
7V
8F
9V
10F

4. COMBINAÇÕES DE AFIRMATIVAS (responda V ou F)

a) (F) 1, 5 e 7 são FALSAS;

b) (F) 1, 5 e 9 são VERDADEIRAS;

c) (V) 2, 5 e 10 são FALSAS;

d) (F) todas as ímpares são VERDADEIRAS;

e) (F) todas as pares são FALSAS;

f) (V) 8 e 10 são FALSAS;

g) (F) somente a 7 é FALSA;

h) (V) 4, 7 e 9 são VERDADEIRAS;

i) (F) somente a 3 e 4 são VERDADEIRAS;

j) (V) 3, 4, 6 e 9 são VERDADEIRAS;

EXERCÍCIO 02

Estruturas de dados lineares, se comparadas entre si, apresentam vantagens e desvantagens ao considerarmos fatores tais como: 1) capacidade da memória do computador; 2) velocidade de processamento; 3) quantidade de elementos a serem armazenados e/ou movimentados; e 4) tempo de acesso a um determinado elemento. Analise as afirmativas apresentadas nas colunas **Vantagem** e **Desvantagem** do quadro abaixo, e estabeleça as quatro combinações que associam corretamente cada estrutura às suas respectivas vantagens e desvantagens.

QUADRO DE VANTAGENS E DESVANTAGENS

Vantagem		Desvantagem		Estrutura	
1	• Não precisa deslocar elementos nas operações de Inserção e Remoção. • Não tem tamanho máximo, o limite é a memória do computador;	A	• Mais memória com UM ponteiro extra	I	Duplamente Encadeada
2	• Acesso direto a qualquer elemento da lista, em tempo constante	B	• Para acessar qualquer elemento é necessário percorrer a lista, até achar o elemento	II	Encadeada Circular
3	• É possível atingir qualquer elemento da lista estando em qualquer nó	C	• Mais memória com ponteiros extras	III	Encadeada
4	• É possível percorrer a lista em ordem inversa; • O nó atual “enxerga” o antecessor e o sucessor; • Último e primeiro nós são equivalentes	D	• Movimentação na inserção/eliminação de um elemento; • Tamanho máximo deve ser pré-definido	IV	Sequencial

Escreva nos espaços as quatro combinações corretas que associam as três colunas da tabela.

Combinação 1) 1 B III

Combinação 2) 2 C I

Combinação 3) 3 D IV

Combinação 4) 4 A II

EXERCÍCIO 03

Com o objetivo de explorar as competências e habilidades para analisar determinada situação/contexto e escolher a estrutura mais adequada para a resolução do problema, utilize as siglas e preencha o espaço de cada contexto descrito de **a)** até **j)**.

LCDE – Lista **C**ircular **D**uplamente **E**ncadeada
LDE – Lista **D**uplamente **E**ncadeada
LE – Lista **E**ncadeada
LS – Lista **S**equencial

a) (LS) Quantidade limitada de elementos.

b) (LS) Sistema com processador lento e necessidade de rápido acesso aos elementos da lista.

c) (LE) Quantidade de elementos aumentará sob demanda.

d) (LE) Muito acesso a elementos não sequenciais.

e) (LDE) Computador com muita memória. Quantidade de elementos indefinida, e necessidade de rápido acesso aos elementos.

f) (LCDE) Necessidade de que do primeiro elemento possa se atingir o último e do último elemento possa se atingir o primeiro. Quantidade de elementos indefinida.

g) (LDE) Necessidade de percorrer a lista para frente e para trás.

h) (LE) Quantidade de elementos indefinida.

i) (LS) Sistema com pouca memória.

j) (LCDE) A lista pode não ter início nem fim, mas crescerá, e precisa-se apenas de um ponteiro para ela.

➢ Considere as seguintes estruturas para as **QUESTÕES 04 E 05**:

```
// Tipo base dos elementos da lista
typedef struct elementos {
    char nome[50];
} t_elemento;

// Estrutura da lista
typedef struct no {
    t_elemento dado;  // elemento contendo os dados
    struct no  * prox; // ponteiro para o proximo elemento
} t_no; // tipo da estrutura

typedef t_no* t_lista; // t_lista é igual a t_no *
```

resposta EX4:

```
void exibirLista (t_lista *lista){
    int i = 0;
    t_elemento *temp = lista;
    while (temp->prox != NULL){
        printf("Nome: %s\n", aluno->nome);
        temp = temp->prox;
        i++;
    }
}
```

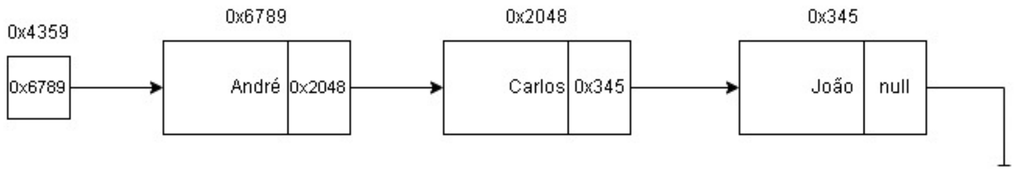
EXERCÍCIO 04

Escreva uma função em C que exiba o dado de todos os nós de uma lista **encadeada**. Esta função receberá como parâmetro a lista, e deverá percorrer toda a lista e ir exibindo a cada nó o valor contido em seu dado. **Não utilize nenhuma outra função auxiliar**.

EXERCÍCIO 05

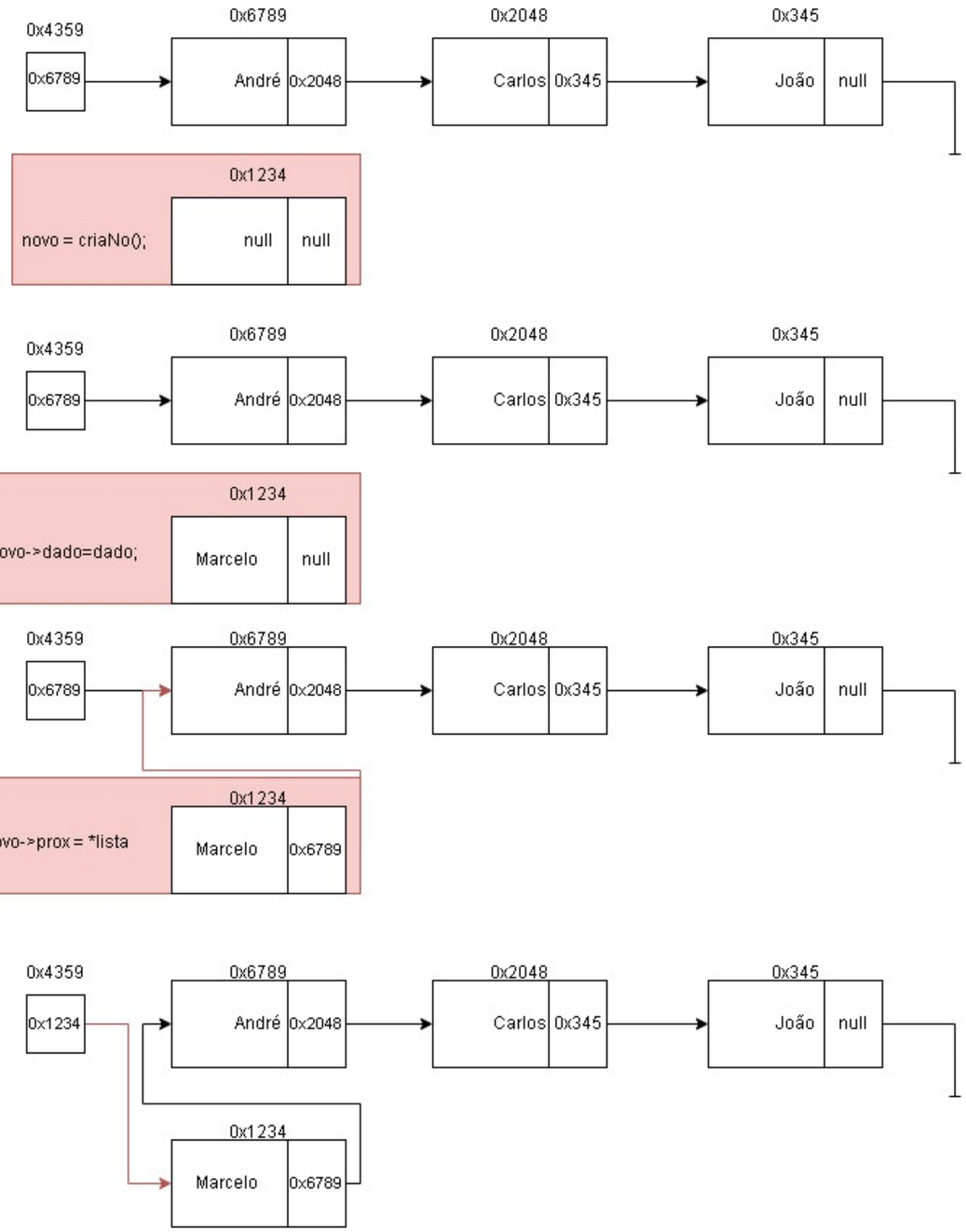
Represente graficamente a execução da função abaixo, considerando uma chamada com pos=0 e outra com pos=2 e considere que a lista inicial tem 3 nós. Lembre-se de indicar todas as variáveis envolvidas, bem como seus respectivos conteúdos e endereços

```
int inserir(t_lista *lista, int pos, t_elemento dado) {
    t_no * p, * novo;
    // inserção na primeira posição
    if (pos == 0) {
        novo = criaNo();
        if (novo == NULL)
            return 0; // erro: memória insuficiente
        novo->dado = dado;
        novo->prox = *lista;
        *lista = novo;
        return 1;
    }
    // inserção após a primeira posição
    p = getNo(*lista, pos-1);
    if (p == NULL)
        return 0; // erro: posição inválida
    novo = criaNo();
    if (novo == NULL)
        return 0; // erro: memória insuficiente
    novo->dado = dado;
    novo->prox = p->prox;
    p->prox = novo;
    return 1;
}
```



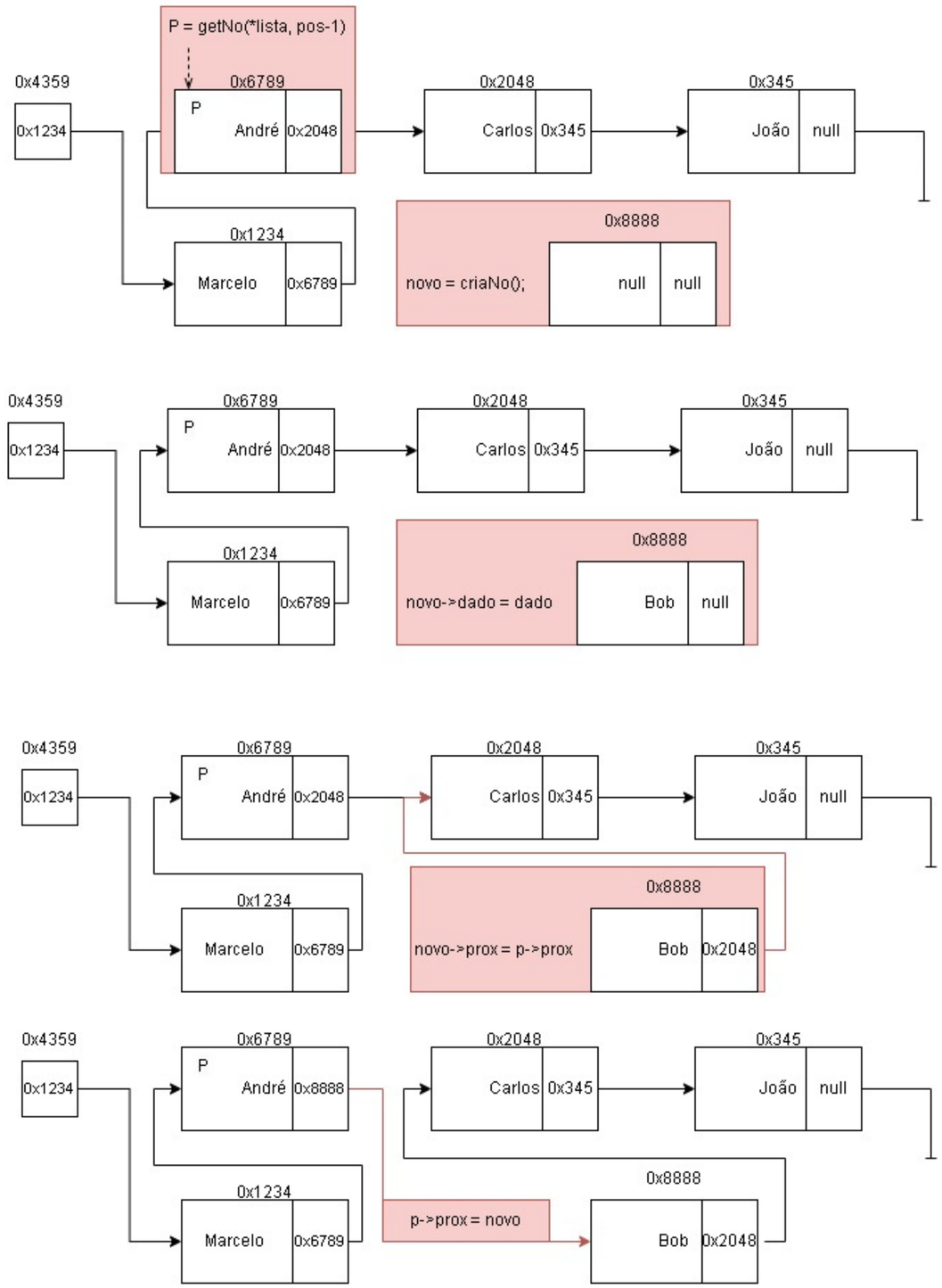
Exemplo1:

inserir(minhaLista, 0, Marcelo)



Exemplo2:

inserir(minhaLista, 2, Bob)



EXERCÍCIO 06

A empresa X de desenvolvimento de software está contratando um profissional para atuar em sua linha de produção de software de alto desempenho. Um dos quesitos do processo de seleção é comprovar competência de analisar e propor melhores soluções de estruturas para manipulação de dados e habilidades para identificar as características do problema a ser solucionado, analisar as estruturas de dados conhecidas e escolher a mais adequada ao problema estudado e propor adequação das estruturas conhecidas para atender à solução do problema real. Desta forma, ela inclui em seus testes a seguinte questão: “Os softwares desenvolvidos pela empresa X manipulam uma grande quantidade de dados (em tempo de execução) para atender a diversas situações e, para tal, utiliza as estruturas **a) listas, b) pilhas e c) filas**, com implementação sequencial e encadeada (dependendo da necessidade da aplicação). Considerando que estas estruturas irão armazenar, em cada nó, o nome e o telefone de uma pessoa, escreva as definições em C (typedef struct) destas três estruturas nas duas implementações”.

```c
// Dados de cada pessoa

typedef struct elemento {
    char[50] nome;
    char[20] telefone;
} dado_pessoa;

// a) Listas: Sequencial e Encadeada


typedef struct letraA_Seq_Lista {
    dado_pessoa dado;
    int tamanho;
} letraA_ListaSequencial;

typedef struct letraA_Enc_No {
    dado_pessoa dado;
    struct letraA_Enc_No * prox;
} LetraA_No_ListaEncadeada;

// b) Pilhas: Sequencial e Encadeada

typedef struct letraB_Pilha_Seq {
    dado_pessoa dado;
    int topo;
} letraB_PilhaSequencial;

typedef struct letraB_Pilha_Enc_No {
    dado_pessoa dado;
    struct letraB_Pilha_Enc_No * prox;
} LetraB_No_PilhaEncadeada;

// c) Filas: Sequencial e Encadeada

typedef struct letraC_Fila_Seq {
    dado_pessoa dado;
    int inicio;
    int fim;
} letraC_PilhaSequencial;

typedef struct letraC_Fila_Enc_No {
    dado_pessoa dado;
    struct letraC_Fila_Enc_No * prox;
} LetraC_No_PilhaEncadeada;

```

EXERCÍCIO 07

Você está na entrada de um beco estreito e sem saída, no qual carros entram e saem em fila única pois não há espaço lateral para dois carros. Neste beco cabem oito carros e você deseja fazer um sistema que controle as entradas, saídas e lotação deste espaço de estacionamento. **Escolha** a estrutura de dados, e correspondente implementação, que melhor se adequa a esta situação e **justifique** tecnicamente sua escolha. **Escreva um algoritmo que simule** as situações abaixo, sendo necessário implementar uma função para cada item (**A** e **C**); os itens **B** e **D** são executados no main() de seu algoritmo:

A) cinco carros entram no beco e seu algoritmo registra (guarda) as placas dos carros que entraram no beco;

B) mostra a placa do quinto carro que entrou no beco;

C) três carros saem do beco e seu algoritmo mostra a placa de cada do carro no momento que ele sai do beco.

D) após a movimentação dos itens **A** e **C**, seu algoritmo mostra quantos carros ficaram no beco

OBS: A imagem abaixo está desatualizada! Na linha 73, só dei pop uma vez, mas na questão são 3 carros que saem, no arquivo da pasta compactada, o código estará atualizado!

OBS2: Na linha 76, imprimo a quantidade de carros chamando a variável topo, porém ela retorna o indice, então, para dizer corretamente a quantidade de carros, devo somar 1 ao seu valor, o código que está na pasta compactada estará atualizado!!

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

typedef struct {
    char placa[10];
} Carro;

typedef struct {
    Carro elemento[8];
    int topo;
} PilhaCarros;

void inicializarPilha(PilhaCarros *pilha) {
    pilha->topo = -1;
}

int estaVazia(PilhaCarros *pilha) {
    return (pilha->topo == -1);
}

int estaCheia(PilhaCarros *pilha) {
    return (pilha->topo == 7);
}

void push(PilhaCarros *pilha, Carro carro) {
    if (estaCheia(pilha)) {
        printf("\nRua esta lotada.\n");
        return;
    }
    pilha->topo++;
    strcpy(pilha->elemento[pilha->topo].placa, carro.placa);
}

Carro pop(PilhaCarros *pilha) {
    Carro carro;
    if (estaVazia(pilha)) {
        printf("\nRua esta vazia.\n");
        strcpy(carro.placa, "");
        return carro;
    }
    carro = pilha->elemento[pilha->topo];
    pilha->topo--;
    return carro;
}

void mostrar(PilhaCarros *pilha){
    int i;
    for (i = 0; i <= pilha->topo; i++){
        printf("\nPlaca do carro %d: %s\n", i+1, pilha->elemento[i].placa);
    }
}

int main() {
    // escolha da estrutura é pilha sequencial,
    // pois só pode sair da rua sem saída, o ultimo carro que entrou
    PilhaCarros minhaPilha;
    inicializarPilha(&minhaPilha);
    char placaTemp[10];
    int i;
    Carro carroTemp;
    // A) cinco carros entram no beco
    for (i = 0; i < 5; i++){
        printf("Digite a placa do carro que entrou: ");
        scanf("%s", placaTemp);
        strcpy(carroTemp.placa, placaTemp);
        push(&minhaPilha, carroTemp);
    }
    mostrar(&minhaPilha);
    // B) mostrar placa do quinto carro
    printf("\n LetraB: Placa do carro 5: %s\n", minhaPilha.elemento[4].placa);
    // C) 3 carros saem do beco e mostra a sua placa
    printf("\n LetraC: Placa do carro que saiu: %s\n", pop(&minhaPilha));
    mostrar(&minhaPilha);
    // D) após A e C, mostrar quantos carros ainda tem
    printf("\n LetraD: No beco, ainda existem %d carros", minhaPilha.topo);
    return 0;
}
```