

1. **(1.0)** Analise o código abaixo.

```
1. int *f1(int *a, int b) {
2.     int *c = a;
3.     for (int i = 0; i < b; i++)
4.         if (a[i] < *c)
5.             c = a + i;
6.     return c;
7. }
8. void f2(int *a, size_t n) {
9.     if (n > 0) {
10.        int *b = f1(a, n);
11.        int c = *a;
12.        *a = *b;
13.        *b = c;
14.        f2(a + 1, n - 1);
15.    }
16. }
17. int main(void) {
18.     int v[] = {5, 1, 3, 5, 10, 2, 4};
19.     f2(v, 7);
20.     for (int i = 0; i < 7; i++)
21.         printf("%d ", v[i]);
22.     return 0;
23. }
```

a. Simule a execução do programa acima e escreva o resultado que será impresso na saída padrão.

1 2 3 4 5 5 10

b. Descreva as operações implementadas pelas funções `f1()` e `f2()`.

A função `f1()` retorna um ponteiro para o menor elemento em um vetor.

A função `f2()` ordena um vetor.

2. **(3.0)** Considerando uma estrutura de nós encadeados (conforme código abaixo), escreva uma função que recebe como entrada um ponteiro para o nó inicial de uma lista de nós encadeados (`cabeca`) e dois inteiros (`a`) e (`b`) e substitua todas as ocorrências de (`a`) na lista pelo valor (`b`). A função deve retornar a quantidade de itens que foram substituídos.

Por exemplo, dada uma lista com os itens [1, 2, 5, 3, 2, 3] e os valores 3 e 6 como parâmetros, a função deve alterar a lista de forma a conter os itens [1, 2, 5, 6, 2, 6] e retornar o valor 2 (quantidade de itens substituídos). Se a mesma lista for passada com os itens 4 e 8, a função deve retornar 0 pois o item 4 não faz parte da lista.

```
typedef struct lista_no ListaNo;
struct lista_no {
    int item;
    ListaNo *prox;
};

int substitui(ListaNo *cabeca, int a, int b) {
    // Seu código
```

```

}

int substitui(ListaNo *cabeca, int a, int b) {
    int n = 0;
    ListaNo *no = cabeca;
    while (no != NULL) {
        if (no->item == a) {
            no->item = b;
            n++;
        }
        no = no->prox;
    }

    return n;
}

```

3. (3.0) A função `sublista()` recebe o ponteiro para o nó inicial de uma lista de nós encadeados e dois inteiros (`inicio`) e (`fim`) e retorna uma nova lista, contendo os elementos localizados entre as posições (`inicio`) e (`fim`) da lista original. Por exemplo, se a função abaixo for aplicada à lista [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], junto com os parâmetros 4 (`inicio`) e 8 (`fim`), deverá retornar uma lista contendo os elementos [5, 6, 7, 8, 9]. No entanto, o código abaixo contém algum erro e não se comporta da forma esperada.

```

1. ListaNo *sublista(ListaNo *cabeca, int inicio, int fim) {
2.     ListaNo *nova_cabeca = NULL, *nova_cauda = NULL;
3.     for (int i = inicio; i <= fim && cabeca != NULL; i++) {
4.         ListaNo *novo = malloc(sizeof(ListaNo));
5.         novo->item = cabeca->item;
6.         novo->prox = NULL;
7.         if (nova_cabeca == NULL)
8.             nova_cabeca = novo;
9.         else
10.            nova_cauda->prox = novo;
11.         nova_cauda = novo;
12.         cabeca = cabeca->prox;
13.     }
14.     return nova_cabeca;
15. }

```

a. Identifique e descreva o erro no código acima.

O código cria uma sublista de tamanho ( $\text{fim} - \text{inicio} + 1$ ) a partir do primeiro elemento da lista e não a partir da posição `inicio` como era esperado.

b. Proponha uma correção para o problema.

Uma solução para o problema é pular os itens até a posição `inicio`. Para isto bastaria inserir entre as linhas 2 e 3 do código acima o seguinte trecho:

```

for (int i = 0; i < inicio && cabeca != NULL; i++)
    cabeca = cabeca->prox;

```

Desta forma a cabeça se deslocaria para posição `inicio` antes de iniciar a criação de novos nós e sua inserção na lista `nova_cabeca` contendo os itens entre as posições `inicio` e `fim`.

c. Qual tipo de operação (criação, produção, modificação, leitura) é implementada por essa função?

Produção, pois gera uma nova lista a partir de uma lista existente

4. (1.5) A função `func()` no código abaixo, utiliza-se das operações básicas de manipulação de pilhas e filas para implementar uma operação em uma `Fila`.

```
1. void func(Fila *f) {
2.     Pilha *p = pilha_nova();
3.     while (!fila_vazia(f))
4.         pilha_insere(p, fila_remove(f));
5.     while (!pilha_vazia(p))
6.         fila_insere(f, pilha_remove(p));
7. }
8. int main(void) {
9.     int dados[] = {1, 2, 3, 2, 1, 2, 3};
10.    Fila *f = fila_nova();
11.    for (int i = 0; i < 7; i++)
12.        fila_insere(f, dados[i]);
13.    func(f);
14.    while (!fila_vazia(f))
15.        printf("%d ", fila_remove(f));
16.    puts("");
17.    return 0;
18. }
```

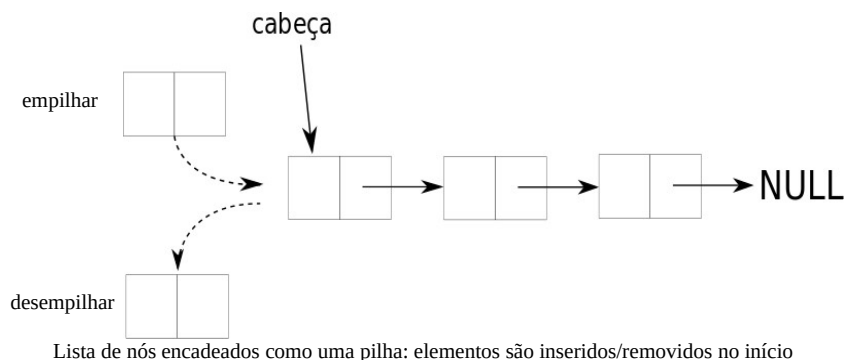
a. Simule a execução do código acima e escreva o impresso na saída padrão.

3 2 1 2 3 2 1

b. Descreva a operação implementada pela função `func()`.

A função **inverte a ordem dos itens em uma fila**

5. (1.0) Uma estrutura de nós encadeados simples pode ser facilmente utilizada para implementar o tipo abstrato de dados pilha, de forma que todas as operações (empilhar, desempilhar, verificar o topo) sejam realizadas em tempo constante (i.e. o número de operações realizadas independe da quantidade de elementos). Isto é possível pois mantemos sempre um ponteiro para o nó cabeça da lista, permitindo que itens sejam inseridos e removidos no início da lista encadeada sem a necessidade de percorrer seus nós.



Esta estrutura de dados também pode ser utilizada, com pequenas variações para implementar o tipo de dados fila. Explique como esta estrutura pode ser utilizada para implementar as operações de uma fila (enfileirar e desenfileirar) em tempo constante.

Para implementar as operações de uma fila utilizando uma estrutura de lista de nós encadeados simples bastaria manter, em adição ao ponteiro para a cabeça (nó inicial), um ponteiro para a cauda (último nó) da lista. Desta forma, é possível remover elementos do início da lista avançando o ponteiro cabeça para o nó seguinte, e adicionar novos elementos ao fim da lista usando o ponteiro próximo do nó cauda. Em ambos os casos não é necessário percorrer os nós da lista.

Não seria necessário, mas caso a explicação seja por meio código/ algoritmo será aceito:

Caso geral, desconsiderando os casos onde a lista está vazia ou possui apenas 1 elemento:

enfileirar(lista, novo\_no)

lista → cauda → prox = novo\_no

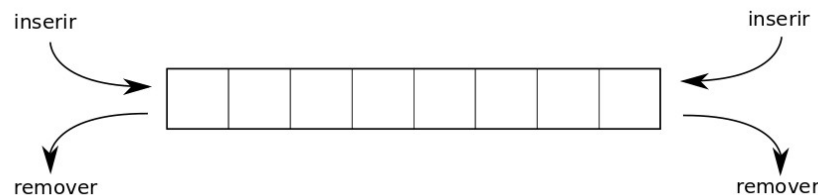
lista → cauda = novo\_no

desenfileirar(lista)

resultado = cabeca

cabeca = cabeca → prox

6. **(1.0)** Uma fila duplamente terminada (*deque*) é um tipo abstrato de dados que representa uma fila na qual é possível a inserir e remover itens tanto em seu início quanto em seu fim. Este tipo de dados pode ser considerado como uma generalização dos tipos pilha e fila pois fornece as operações de ambos.



Descreva quais estruturas de dados podem ser utilizadas par implementar um *deque*, de forma que as operações de modificação (inserir/remover no início/fim) possam ser executadas em tempo constante (sem a necessidade de percorrer os itens). Justifique sua resposta, explicando porque essa estrutura atende essa condição.

1. Utilizando uma lista duplamente encadeada. Uma lista duplamente encadeada permite percorrer os elementos nas duas direções, possibilitando inserir e remover elementos em ambas as extremidades da lista, desde que mantidos ponteiros para a cabeça e cauda.

2. Utilizando um vetor o tempo será constante para os casos onde temos uma fila de tamanho máximo fixo. Caso a fila não tenha um tamanho máximo fixo, o tempo será constante amortizado, pois na maior parte das vezes as operações são em tempo constantes, mas ocasionalmente redimensionamentos podem ser necessários. Também será considerada correta a resposta utilizando um vetor.