

1. (1.0) Analise o código abaixo.

```
1. void calc(double *a, size_t n, double *c, double *d) {
2.     double e = 0, f = 0;
3.     for (int i = 0; i < n; i++) {
4.         e += a[i];
5.         f += 1;
6.     }
7.     *c = e;
8.     *d = e / n;
9. }
10. int main() {
11.     double a = 10, b = 3, v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
12.     calc(v + 3, 5, &a, &b);
13.     printf("%.1lf %.1lf\n", a, b);
14.     return 0;
15. }
```

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

b. Descreva com suas palavras a operação implementada pela função `calc()`.

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 um inteiro (`item`), retorne a posição da primeira ocorrência do item na lista. A função deve retornar o valor `-1` caso o item não seja encontrado na lista.

Por exemplo, dada uma lista com os valores [4, 2, 5, 3, 2, 3] e o item 2, a função deve retornar a posição 1, pois a primeira ocorrência do item 2 é segunda posição da lista.

```
typedef struct no No;
struct no {
    int item;
    No *prox;
};

int posicao(No *cabeca, int item) {
    // Seu código
}
```

4. (1.5) Identifique qual tipo abstrato de dados pode ser utilizado para implementar os botões voltar e avançar em um navegador *web*. Explique a ideia da solução, descrevendo o que deve ser feito em cada uma das três situações seguintes: o usuário clica no botão voltar; o usuário clica no botão avançar; ou o usuário digita um novo endereço.

3. (3.0) A função `remove_todos()` deveria remover todas as ocorrências de um determinado item em uma lista iniciada no nó `cabeca`. No entanto, o código abaixo contém um erro de lógica, fazendo com que a função não apresente o comportamento esperado.

```

1.  /* Remove todas as ocorrências de item na lista */
2.  No *remove_todos(No *cabeca, int item) {
3.      while (cabeca != NULL && cabeca->item == item) {
4.          No *temp = cabeca;
5.          cabeca = cabeca->prox;
6.          free(temp);
7.      }
8.      if (cabeca != NULL) {
9.          No *anterior = cabeca;
10.         No *proximo = cabeca->prox;
11.         while (proximo != NULL) {
12.             if (proximo->item == item) {
13.                 No *temp = proximo;
14.                 anterior->prox = proximo->prox;
15.                 free(temp);
16.             }
17.             proximo = anterior->prox;
18.         }
19.     }
20. }
21. return cabeca;
22. }

```

a. Identifique o erro no código acima. Simule a execução em uma entrada e descreva qual o comportamento da função.

b. Proponha uma correção para o problema.

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

5. (1.5) A função `func()` no código abaixo, utiliza-se das operações básicas de manipulação de pilha (empilhar, desempilhar e verificar se a pilha está vazia) para implementar uma nova operação para este tipo dados. Você deve considerar que o tipo de dados `Pilha` e as funções `vazia()`, `desempilha()` e `empilha()` já foram implementadas.

```

1.  void func(Pilha *pilha, int x) {
2.      if (!vazia(pilha)) {
3.          int topo = desempilha(pilha);
4.          func(pilha, x);
5.          if (topo != x)
6.              empilha(pilha, topo);
7.      }
8.  }
9.  int main(void) {
10.     int dados[] = {1, 2, 3, 2, 1, 2, 3};
11.     Pilha *p = pilha_nova();
12.     for (int i = 0; i < 7; i++)
13.         empilha(p, dados[i]);
14.     func(p, 3);
15.     while (!vazia(p))
16.         printf("%d ", desempilha(p));
17.     puts("");
18.     return 0;
19. }

```

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

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