Universidade Federal de Goiás
Engenharia da Computação
Estruturas de Dados 1

1ª Avaliação – 2019.1

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.

R: O programa imprime os valores 30.0 e 6.0.

É passado para a função calc um ponteiro para o 4º elemento do vetor v (v + 3) e n = 5. Portanto é passado o subvetor contendo os itens da posição 4 à 8 (4, 5, 6, 7, 8). A soma desses números é 30 e a média aritmética 6.

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

R: A função calc() calcula a soma e média aritmética dos números em um vetor de pontos flutuantes de dupla precisão.

A função recebe como argumentos um ponteiro para um vetor de sflutuantes de dupla precisão (a) e o tamanho do vetor (n) e retorna a soma e a média dos números no vetor por meio dos ponteiros (c) e (d). Os ponteiros são utilizados como forma de passagem por referência, permitindo à função retornar mais de um valor.

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;
};
```

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

R:

```c
int posicao(No *cabeca, int item) {
    int i = 0;
    No *no = cabeca;
    while (no != NULL) {
        if (no->item == item)
            return i;
        i++;
        no = no->prox;
    }
    return -1;
}
```

Outras soluções também são possíveis.

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.

R: Para implementar essa funcionalidade podemos utilizar duas pilhas. A primeira pilha mantém um histórico de endereços que podem ser acessados por meio do botão voltar. A segunda pilha contém uma lista de endereços para os quais o usuário pode avançar.

Quando o usuário clica no botão voltar, o último endereço acessado anteriormente é desempilhado da primeira pilha e o navegador abre a página correspondente. Além disso, o endereço da página aberta no momento do clique do botão voltar é empilhado na segunda pilha, para que o usuário possa posteriormente avançar para este endereço.

Quando o usuário clica no botão avançar a operação inversa é executada. O endereço no topo da segunda pilha é desempilhado e o navegador abre a página correspondente. O endereço anterior é empilhado na primeira pilha.

Quando o usuário digita um novo endereço, o endereço anterior é empilhado na primeira pilha. Além disso, a segunda pilha é esvaziada, pois só é possível avançar após uma operação voltar.

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.

```c
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.
18.                 proximo = anterior->prox;
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.

R: A função entra em repetição infinita quando encontra um nó que não contém o valor a ser removido.

A cada iteração do laço de repetição entre as linhas 11 e 20, a função verifica se o nó seguinte (proximo) contém o valor a ser removido e corretamente remove este item (linhas 12 à 16). O ponteiro para o nó anterior continua apontando para o mesmo nó e o ponteiro proximo é atualizado para o nó seguinte (linha 18). No entanto, caso em que o nó proximo não contém o valor a ser removido o ponteiro anterior também continua inalterado, fazendo com que a função entre uma repetição infinita.

b. Proponha uma correção para o problema.

R: Uma solucão para o problema identificado é atualizar o ponteiro anterior, no caso em que o nó proximo não contém o item a ser removido. Ou seja, o seguinte código pode ser inserido entre as linhas 16 e 17:

```
} else {
    anterior = proximo;
}
```

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

R: Modificação

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.

R: O programa imprime os valores 2 1 2 2 1

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

R: A função remove todas as ocorrências de x na pilha.

A função utiliza recursão para executar essa operação. Um item é retirado do topo (linha 3) e descartado caso seja igual ao parâmetro x, caso contrário o item é reempilhado (linhas 5 e 6). Antes de reempilhar o item a função é reaplicada no restante da pilha (linha 4). Desta forma a função é executada sucessivas vezes, em cada execução descartando o item no topo caso este seja igual à x.