


Estruturas De Dados

Respostas da Atividade Especial

1. Analise os seguintes algoritmos e determine sua complexidade (constante ou linear): 

a)

```
int main()
{
    int quantas_vezes;
    scanf("%d", &quantas_vezes);


    for(int i = 1; i <= quantas_vezes; i++){
        printf("É melhor ter duas pedras no caminho do que uma nos rins, hidrate-se\n");
    }
}
```

b)

```
printf("- Mó ruim C, ninguém usa. Muito melhor Python\n");
printf("Fez-se um silêncio constrangedor na sala. Até que alguém finalmente se levanta:");
printf("- Escuta aqui meu consagrado, você respeite o C.\n");
```

2. Girão possui um irmão mais novo o qual quis seguir seus passos e, também, está cursando Ciência da Computação. Agora, no seu segundo semestre, pediu para o irmão (Girão) ensiná-lo a implementar uma lista encadeada. Girão, como um bom irmão, decidiu ajudar seu caçula nesta tarefa. A dúvida do irmão é como implementar a função adicionar na lista encadeada.

Agora, você fará o papel de Girão! De acordo com seus conhecimentos adquiridos no curso de Estrutura de Dados, implemente e explique, da sua maneira (escrevendo e/ou desenhando), a função adicionar de uma lista encadeada.

3  Depois de explicar como implementar a função adicionar de uma lista encadeada, o irmão de Girão quer aprender, agora, como fazer a função remover. Com o mesmo molde da 5ª questão, implemente e explique a função remover de uma lista encadeada.

4. Após uma das aulas de Estrutura de Dados, seu colega de sala pede a sua ajuda na implementação das funções adicionar e remover em uma lista encadeada com ponteiro no início e fim.

```
typedef struct personagem{  
    char nome[80];  
    char historia[120];  
    struct personagem *prox;  
}Personagem;
```

Tendo isso em vista a **struct** acima, complete, substitua ou remova corretamente as funções adicionar e remover.

a) **adicionar()**

```

16 void adicionar(char *name, char *history, int pos){
17     if(pos >= 0 && pos <= tam){
18
19         Personagem ... = malloc(sizeof(...));
20         Personagem->nome = ...;
21         ...->historia = history;
22
23         if(inicio == NULL){
24
25             inicio = Novo;
26             fim = ...;
27         }else if(pos == ...){
28
29             Novo->prox = inicio;
30             inicio = Novo;
31         }else if(pos == tam){
32
33             fim->... = Novo;
34             fim = Novo;
35         }else{
36
37             NO *aux = inicio;
38
39             for(int i = 0; i < pos-1; i ++){
40                 aux = aux->prox;
41             }
42
43             Novo->prox = aux->...;
44             aux->prox = Novo;
45         }
46         ...
47     }else{
48         printf("Posicao Invalida!");
49     }
50 }

```

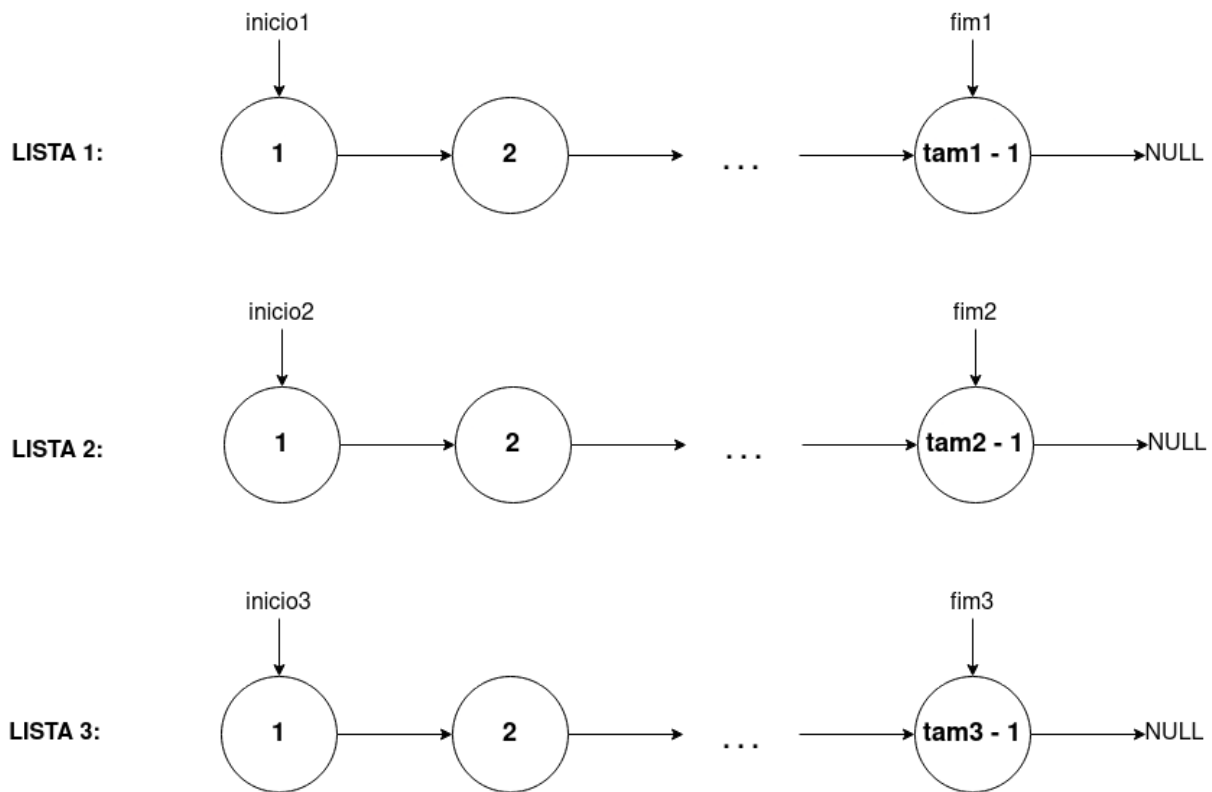
b) remover()

```

void remover(int pos){
    if(pos >= 0 && pos <= tam){
        if(tam == 1){
            ... *lixo = inicio;
            inicio = NULL;
            fim = NULL;
            free(...);
        }else if(pos == 0){
            Personagem *lixo = inicio;
            inicio = inicio->...;
            inicio->... = NULL;
            free(lixo);
        }else if(pos == tam-1){
            Personagem *lixo = ...;
            Personagem *aux = inicio;
            for(int i = 0; i < tam-1; ++i){
                aux = aux->prox;
            }
            aux->prox = NULL;
            fim = ...;
            free(lixo);
        }else{
            Personagem *lixo = fim;
            ... *aux = inicio;
            for(int i = 0; i < pos-1; ++i){
                ... = aux->prox;
            }
            aux->prox = aux->...->...;
            aux->prox->... = NULL;
            free(lixo);
        }
        tam--;
    }else{
        printf("Posicao Invalida!");
    }
}

```

5. Implemente 3 listas encadeadas com ponteiro no início e no fim, paralelas, no mesmo programa. Cada uma deve ter suas variáveis início, fim e tamanho. Como exemplificado abaixo:



Seu programa deve ter as funções de adicionar, remover e imprimir elementos.

:D

6. Você conhece a função `rand()` da linguagem C? Ela é uma função que pode ser importada da biblioteca-padrão de utilitários gerais `<stdlib.h>`, e é usada para gerar números aleatórios! A maneira de utilizá-la é bem simples, ela pode ser chamada sem nenhum argumento e retorna um número inteiro positivo pseudo aleatório (Basicamente um número quase aleatório).

```
int x = rand(); // x recebe um valor aleatório
```

“Ah mas e se eu quiser gerar um número em um intervalo específico, como é que eu faço”, você deve estar se perguntando, mas é bem simples caro ser humano, lembra do operador `%` (módulo), usado pra calcular o resto da divisão de um número por outro? basta fazer assim:

```
int x = rand() % 10; // x recebe um valor aleatório de 0 a 9
```

Assim você consegue gerar números de 0 a 9. Caso você queira números de 0 a 100, por exemplo, é só colocar lá 101 no lugar do 10, entendeu?

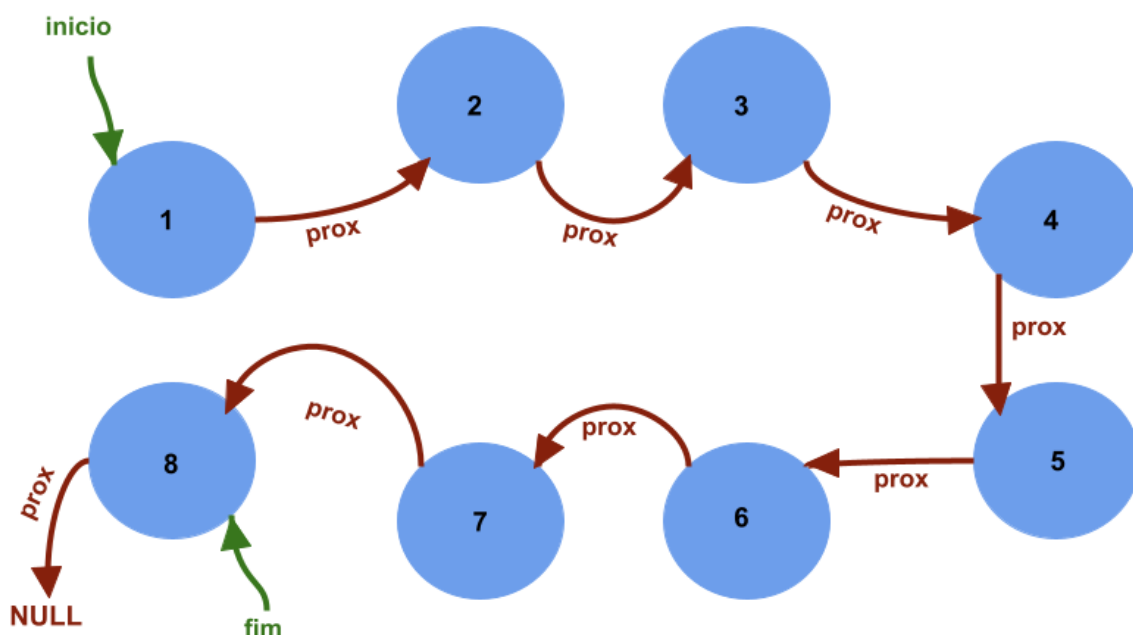
Agora sem mais delongas, vamos ao enunciado de verdade da questão.

Usando a função `rand()` ao seu favor, crie uma lista encadeada com ponteiro no fim, que possua uma função de adicionar um valor inteiro aleatório em uma posição da lista aleatória.

7. Sabemos que há mais de uma forma de fazer uma lista encadeada. Nas aulas vimos duas delas, uma com e outra sem o ponteiro final, em que esse serve para diminuir a complexidade em fazer certas operações no final da lista. Sabendo que, fazendo certas modificações na lista encadeada, também é possível remover um nó que está exatamente no meio da lista de forma menos complexa do que foi ensinado em aula, **explique** quais modificações você faria na lista em si e na função de remover para remover esse nó de forma **constante** (considere que o elemento central de listas com um número de elementos par é aquele que está o mais próximo do centro possível, mas está mais um pouco mais distante do final do que do início dela).

8. Após realizar a questão anterior você deve ter percebido que, para remover o nó central de forma constante, **algo** de fácil acesso deve sempre estar referenciando esse nó ou referenciando um nó próximo a ele, caso contrário, não é possível chegar ao elemento que desejamos remover. Utilizando esse algo (que nesse ponto você já deve saber o que é), altere a operação de adicionar um nó na lista fazendo com que ela agora possua outro caso de adição. Esse caso é: **adicionar em uma posição que fica após o meio da lista**. Ele deve ser menos complexo do que o caso em que adicionamos um nó no meio de uma lista encadeada e não deve realizar a operação de adicionar no fim da lista. Note que não há uma forma simples de adicionar um elemento de forma constante no meio de uma lista (isto é, se é que existe essa opção). Logo, esse novo caso da função adicionar não deve ser constante, mas deve realizar um número menor de operações comparado ao caso comum de adicionar um nó em qualquer local no meio da lista.

9. Agora que já sabemos implementar uma **lista encadeada com ponteiro no fim**, podemos pensar um pouco mais sobre como utilizar ele em outros tipos de listas. Em um playlist de músicas no spotify, por exemplo, temos uma opção de **repetir** a playlist sempre que todas as músicas dela forem tocadas. Podemos pensar nisso como uma lista encadeada circular, onde o último e o primeiro nó da lista estão ligados e assim nossa lista se torna um “círculo”, pois não tem exatamente um “fim”. Veja o desenho abaixo e responda, **qual operação** devemos acrescentar nos códigos de lista com ponteiro no fim para transformar em uma lista circular?



10. No jogo League of Legends, onde se é mais conhecido como LOL. Tem se uma formação de grupo com **5 personagens**, e cada personagem tem sua **rota** dependendo do posicionamento que irá jogar. As rotas, que são mais conhecidas como lanes, são nomeadas de: **TOP, MID, JUNGLE, SUP** e **ADC**. Abstraindo essas regras, o código abaixo é uma lista encadeada, que segue com uma ordem de ligações de acordo com as lanes e o personagem de cada uma delas. Porém, estão faltando alguns trechos de código para a formação correta do grupo em questão. **Complete os trechos** que faltam para executar o código e, em seguida, **desenhe o passo a passo** da formação da lista, de acordo com o que foi visto em aulas.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct no{
6      char campeao[10];
7      struct no *rota;
8  } NO;
9
10 void formacao(){
11
12     //////////// PASSO 1 ////////////
13     NO *jungle = malloc(sizeof(NO));
14     NO *mid = malloc(sizeof(NO));
15     NO *adc = malloc(sizeof(NO));
16     NO *sup = malloc(sizeof(NO));
17     NO *top = malloc(sizeof(NO));
18
19     //////////// PASSO 2 ////////////
20     strcpy(jungle ... campeao, " ... ");
21     strcpy(mid-> ... , "gwen");
22     strcpy( ... -> ... , "ashe");
23     strcpy(sup->campeao, " ... ");
24     strcpy(top->... , "darius");
25
26     top-> ... = mid; //////////// PASSO 3.1 ////////////
27     ... ->rota = jungle; //////////// PASSO 3.2 ////////////
28     jungle ... rota = ... ; //////////// PASSO 3.3 ////////////
29     sup->rota = adc; //////////// PASSO 3.4 ////////////
30
31     //////////// PASSO 4 ////////////
32     printf("\n");
33     printf("|TOP: %s| -> ", ... );
34     printf("|MID: %s| -> ", ... );
35     printf("|JUNGLE: %s| -> ", ... );
36     printf("|SUP: %s| -> ", ... );
37     printf("|ADC: %s|", ... );
38 }
39
40 int main(void){
41     formacao();
42 }

```

Saída do código:

```
|TOP: darius| -> |MID: gwen| -> |JUNGLE: evelynn| -> |SUP: morgana| -> |ADC: ashe|
```