Aula 05 - Estrutura de dados 1 – Uniube

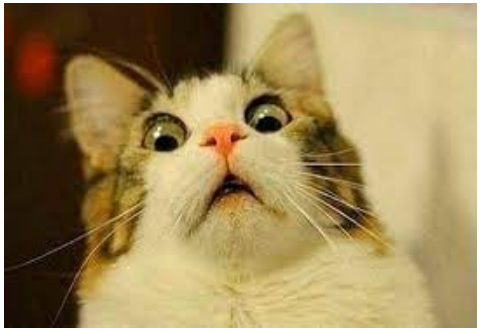
Prof. Marcos Lopes

34 9 9878 0925

Inserção em uma lista encadeada (sem cabeça)

A inserção numa lista encadeada sem a célula da cabeça exige o uso de ponteiro de ponteiro. Além disso, de modo geral a lista trabalha com inserções no final, vamos analisar a função insere a seguir:

```
void insere (int x, celula **p){
   celula *nova;
   nova = (celula*) malloc (sizeof (celula));
   nova->conteudo = x;
   if(*p == NULL) {
      *p = nova;
      return;
   }
   celula *current = *p;
   while(current->prox != NULL) {
      current = current->prox;
   }
   current->prox = nova;
}
```



Exercícios:

a) Desenhe um diagrama esquemático da memória durante a operação de inserção de um elemento na lista vazia (usando a função do slide anterior).

b) Por que é ncessário o conceito de ponteiro de ponteiro?

c) Desenhe um diagrama esquemático da memória durante a operação de inserção de um elemento na lista não vazia (usando a função do slide anterior).

d) Descreva um algoritmo representando a função insere do slide anterior.

Remoçao em uma lista encadeada

Considere o problema de remover uma certa célula de uma lista encadeada.

Como remover no inicio da lista? Como encontrar a célula a swer removida? Como remover a célula quando encontrada? E se não for encontrada? Como liberar a memória?

Essas questões devem ser consideradas na operação de remoção de elementos de uma lista encadeada.

Vamos ver um algoritmo básico pra essa operação de remoção definida como:

```
void remove(int x, celula **p);
```

Algoritmo:

1) verifica se a primeira célula não é nula e se tem o conteúdo
    - se sim, aponta a lista pra proxima célula e libera a primeira (free)
    - retorna

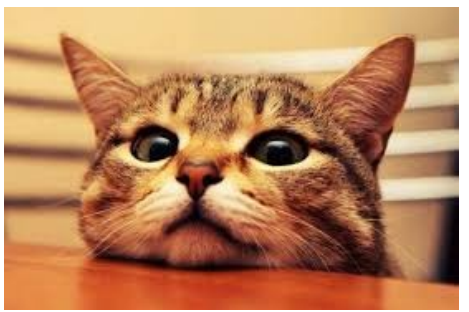
2) percorre todas as células (while) até bater em NULL ou encontrar o conteúdo
    - mantem a referencia da célula anterior (prev) e da atual (current) durante o processo

3) após sair do laço ver se a célula corrente é NULL
    - se for não encontrou o elemento
    - retorna (sem remover nada)

4) se a célula atual (current) é diferente de NULL, achou
    - atribui a prox (da celula anterior) o valor de prox da celula corrente
    - libera a memória da célula corrente (free)

Exemplo de implementação:

```
void remove(int x, celula **p) {
    struct celula* current = *p;
    struct celula* prev = NULL;
    if (current != NULL && current->conteudo == x) {
        *p = current->prox;
        free(current);
        return;
    }
    while (current != NULL && current->conteudo != x) {
        prev = current;
        current = current->prox;
    }
    if (current == NULL) {
        //printf("Elemento %d não encontrado na lista.\n", target);
        return;
    }
    prev->prox = current->prox;
    free(current);
}
```



Exercícios:

a) Desenhe um diagrama esquemático da memória durante a operação de remoção de um elemento (encontrado) na lista não vazia (usando a função do slide anterior).

b) Teste a função pra vários cenários: remoção do primeiro elemento, remoção do último elemento, remoção de um elemento no ‘meio’ da lista, remoção de um elemento não existente na lista.