Avaliação 1 -

• Discente : Paulo Henrique Diniz de Lima Alencar. • Ciência da Computação

1 - Resolução :

a) Resolução :

* Código :

```c
#include <stdlib.h>
#include <stdio.h>

struct no {
    int id;
    int sexo;
    int num_filhotes;
    struct no *prox;
    struct no *ant;
};

struct no typedef NO;
```

arquivo No.h

* Justificativa : na definição da minha struct do tipo NO foi necessário adicionar os seguintes atributos :

* Id - inteiro - identifica cada canguru ;

* Sexo - inteiro - armazena 1 se canguru fêmea, 0 se canguru macho ;

* num_filhotes - inteiro - representa a quantidade de filhotes ;

* struct no *ant - ponteiro do tipo struct no responsável por apontar para o Nó anterior.

*OBS: struct no *Prox ; já presente na definição, portanto não houve necessidade de adicionar.

---

b) Resolução :

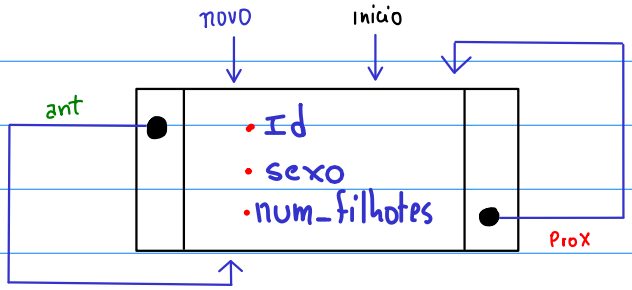
* Código :

```c
// Discente: Paulo Henrique Diniz de Lima Alencar.
// Matricula 494837


// Incluindo bibliotecas em meu header (cabeçalho)
#include <stdlib.h>
#include <stdio.h>
#include "No.h"


// Variáveis GLOBAIS
NO *inicio;
int tam = 0;


// Procedimento: responsável por adicionar cangurus em uma lista duplamente encadeada circular.
void add (int id, int sexo, int num_filhotes) {
    NO *novo = malloc (sizeof(NO));
    novo -> id = id;
    novo -> sexo = sexo;
    novo -> num_filhotes = num_filhotes;

    if (tam == 0) {
        novo -> prox = novo;
        novo -> ant =  novo;
        inicio = novo;
    }
    else {

        novo -> prox = inicio;
        novo -> ant = inicio -> ant;
        inicio -> ant -> prox = novo;
        inicio -> ant = novo;

    }
    tam++;
}
```

arquivo ListaCircular.c

* Justificativa:

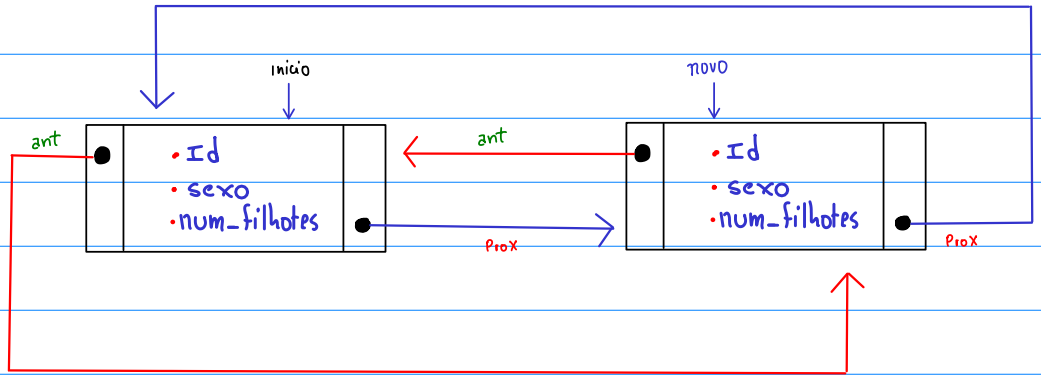
• Quando minha lista está vazia, isto é, tam == 0, e adiciono o 1º Nó:



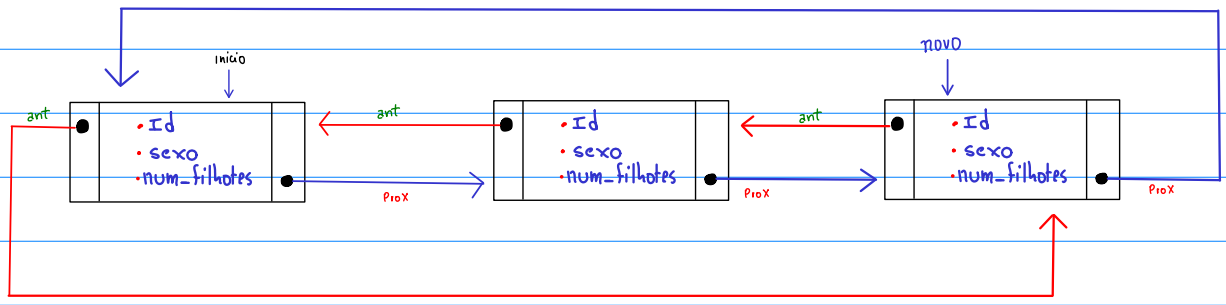

```
if (tam == 0) {
    novo -> prox = novo;
    novo -> ant =  novo;
    inicio = novo;
}
```

arquivo ListaCircular.c

• cai no else:

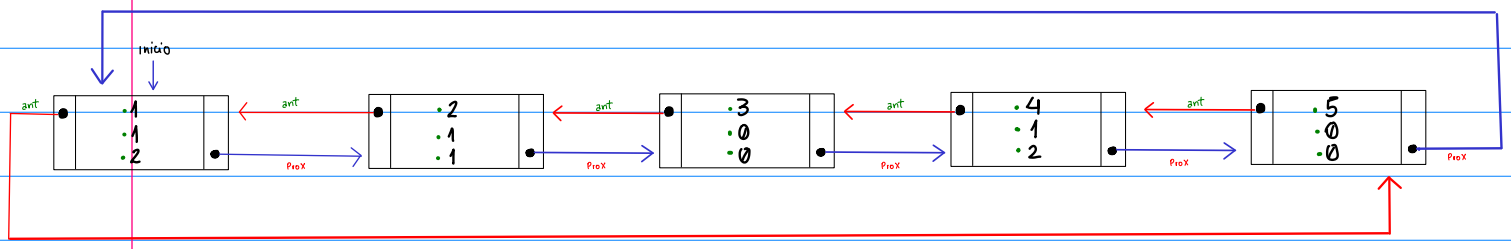


arquivo ListaCircular.c



arquivo ListaCircular.c

* LDE circular após a execução do código:



c) Resolução:

* Código:

```
65 int soma () {
66      NO* i = inicio;
67
68      int soma = 0;
69      do {
70         if (i -> sexo == 1) {
71              soma = soma + i -> num_filhotes;
72         }
73         i = i -> prox;
74      } while (i != inicio);
75
76      return soma;
77 }
78
```

arquivo ListaCircular.c

* Justificativa:

- funcionamento

percorre cada No da LDE circular, verificando se o canguru é fêmea. Se é fêmea adiciona na variável soma a quantidade de filhotes que ela possuí. Essas verificações são feitas até o ponteiro i deter novamente o endereço do ponteiro início da lista (isso significa que a lista já realizou um ciclo completo). Por fim, retorna a soma (soma = 5 na execução do código).

2 - Resolução:

a) Resolução:

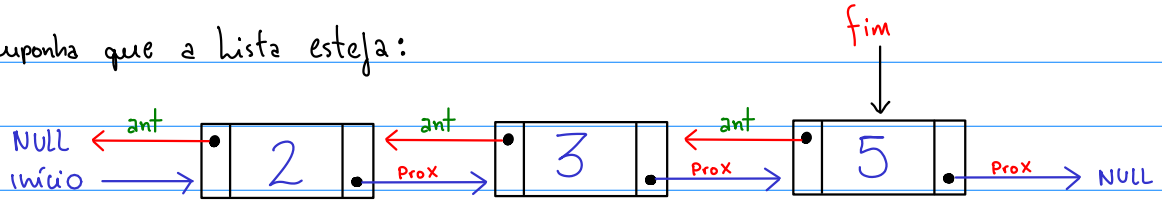
- Vou desenhar a remoção de um nó em uma posição n qualquer, tal que, n > 0 e n < tamanho da lista.

* Código completo:

```
54 // Função: responsável por remover um Nó em uma posição n qualquer, tal que, n > 0 e n < tamanho da lista.
55 int remover (int n) {
56
57     int removido = -1;
58
59     if (n > 0 && n < tamanho) {
60         No* lixo;
61
62         if (n == tamanho - 1) { //.... Remover elemento na última posição.
63             lixo = fim;
64             fim = fim -> ant;
65             fim -> prox = NULL;
66         }
67         else { //..................... Remover elemento no "meio"
68             No* aux = inicio;
69             int i;
70             for (i = 0; i < n; i++) {
71                 aux = aux -> prox;
72             }
73             lixo = aux;
74             aux -> prox -> ant = aux -> ant;
75             aux -> ant -> prox = aux -> prox;
76         }
77         removido = lixo -> valor;
78         free(lixo);
79     tamanho--;
80     }
81     else {
82         printf("Posição inválida !! \n");
83     }
84     return removido;
85 }
```

Remover quando n == tamanho - 1, isto é, remover último Nó:

- Suponha que a lista esteja:




```
if (n == tamanho - 1) { //.... Remover elemento na última posição.
    lixo = fim;
    fim = fim -> ant;
    fim -> prox = NULL;
}
```

- Primeiramente colocamos o ponteiro lixo para apontar para o Nó que desejamos remover. Logo, precisamos executar a seguinte operação: lixo = fim ;

- Como estamos removendo o último Nó, o último agora passa a ser o Nó anterior. Para isso acontecer executamos a seguinte operação: fim = fim → ant ;

- Em seguida, precisamos fazer com que o fim → prox aponte para NULL, indicando o fim de nossa lista. Para isso precisamos realiza a seguinte operação: fim → prox = NULL;

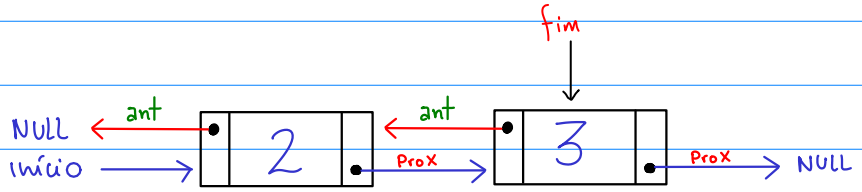
• Por fim, precisamos precisamos desalocar a porção de memória que foi alocado por malloc. Assim, free (lixo) vai informar ao S.O que o bloco de bytes apontado por lixo, está livre para reciclagem. É isso que o trecho abaixo faz.

free

```
79      free(lixo);
79  tamanho--;
80  }
```

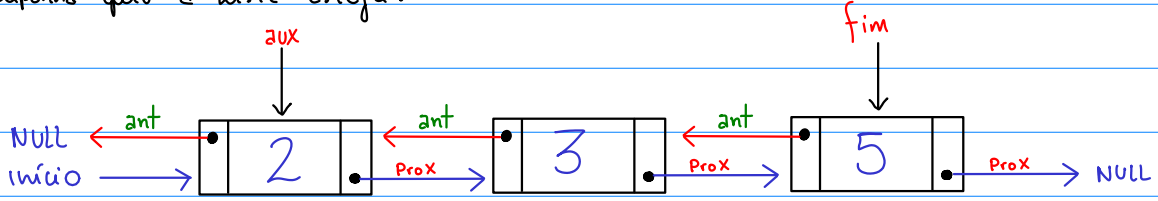
precisamos também reduzir em 1 unidade o tamanho de nossa LDE.

A lista vai ficar:



☑ Remover no "meio":

• Suponha que a lista esteja:

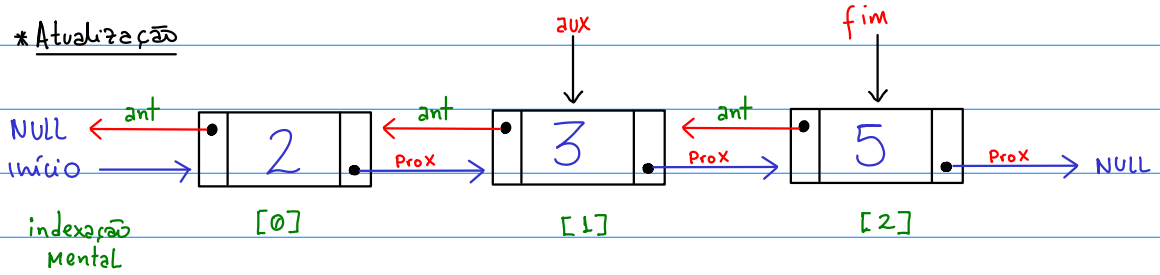



```
    else { //..................... Remover elemento no "meio"
        No* aux = inicio;
        int i;
        for (i = 0; i < n; i++) {
            aux = aux -> prox;
        }
        lixo = aux;
        aux -> prox -> ant = aux -> ant;
        aux -> ant -> prox = aux -> prox;
    }
    removido = lixo -> valor;
    free(lixo);
tamanho--;
}
```

• Primeiramente precisamos percorrer com o ponteiro aux até o nó que desejamos remover. Para fazer isso é necessário fazer uso de uma estrutura de repetição, mas especificamente um for, que vai permitir percorrermos do início da nossa lista, até o Nó à ser removido. Operações necessárias:

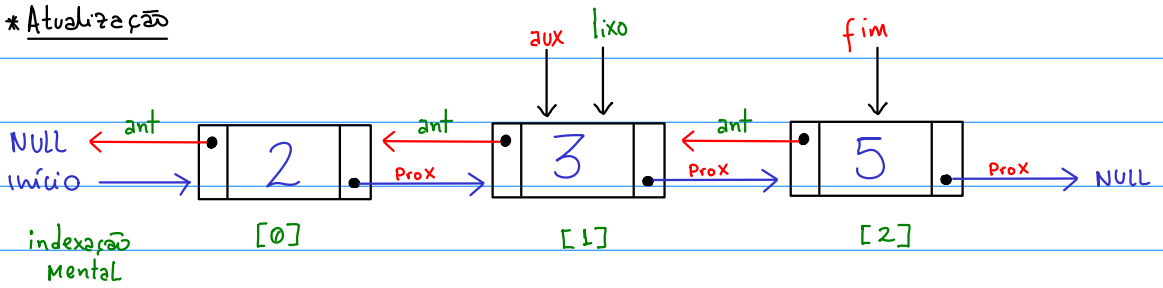
```
No* aux = inicio;
int i;
for (i = 0; i < n; i++) {
    aux = aux -> prox;
}
```

Suponha que a posição que desejamos remover é n = 2, nosso aux ficá no seguinte nó:



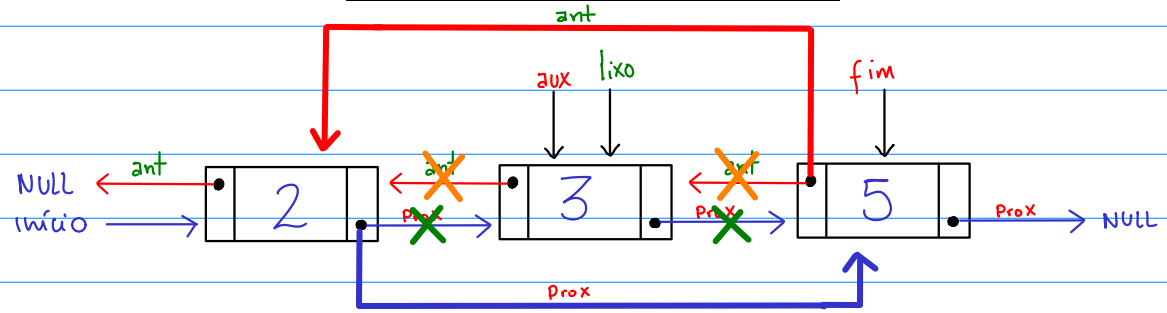
- Em seguida, colocamos o ponteiro lixo para apontar para o Nó que desejamos remover. Logo, precisamos executar a seguinte operação: lixo = aux;

```
No* aux = inicio;
int i;
for (i = 0; i < n; i++) {
    aux = aux -> prox;
}
lixo = aux;
aux -> prox -> ant = aux -> ant;
aux -> ant -> prox = aux -> prox;
```

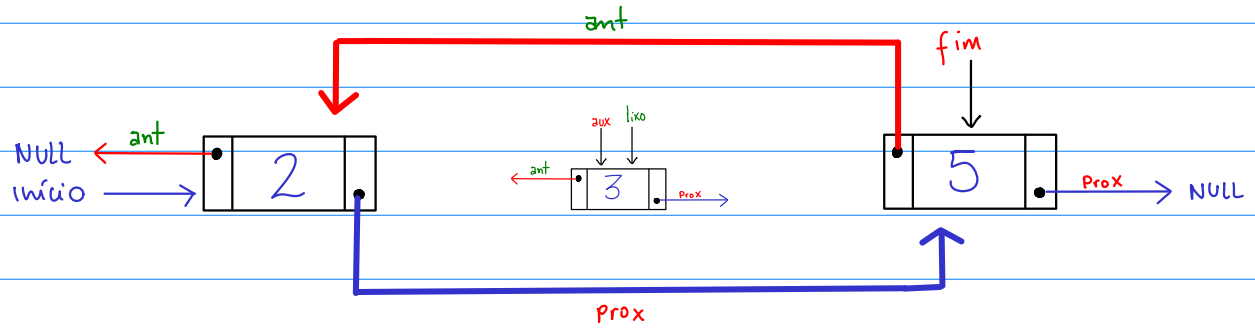


- Após isso precisamos realizar determinadas operações para que no momento que removermos o Nó desejado, não gere buracos em nossa lista.

```
No* aux = inicio;
int i;
for (i = 0; i < n; i++) {
    aux = aux -> prox;
}
lixo = aux;
aux -> prox -> ant = aux -> ant;
aux -> ant -> prox = aux -> prox;
```



* Atualização



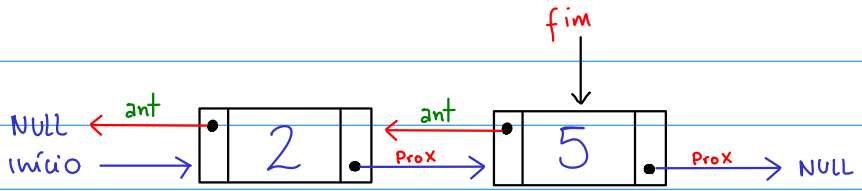
• Por fim, precisamos precisamos desalocar a porção de memória que foi alocado por malloc. Assim, free (lixo) vai informar ao S.O que o bloco de bytes apontado por lixo, está livre para reciclagem. É isso que o trecho abaixo faz.

free

```
78          free(lixo);
79      tamanho--;
80      }
```

precisamos também reduzir em 1 unidade o tamanho de nossa LDE.

A Lista vai ficar:



---

b) Resolução:

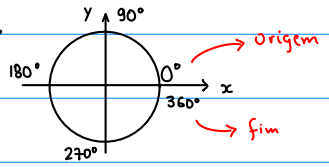
▨ Lista implementada com VETOR:

* Adicionar no início - O(n) - Linear.
* Adicionar no fim - O(1) - Constante.
* Adicionar na n-ésima posição - O(n) - Linear.
* Remover do início - O(n) - Linear.
* Remover do fim - O(1) - constante.
* Remover da n-ésima posição - O(n) - Linear.

Lista da Questão 1:

* Adicionar no início - O(1) - constante.
* Adicionar no fim - O(1) - constante.
* Adicionar na n-ésima posição - O(n) - Linear.
* Remover do início - O(1) - constante.
* Remover no fim - O(1) - constante.
* Remover na n-ésima posição - O(n) - linear.

Obs: Como estou trabalhando com uma LDE circular, não há o conceito de posição. Porém, lidando de forma abstrata posso considerar que minha LDE circular possui um início, que será o Nó que o ponteiro início aponta. Esse Nó pode representar de maneira abstrata o início do meu "ciclo". Essa ideia é semelhante a um círculo trigonométrico, que em tese não possui início, nem fim. No entanto, existem pontos padrões que facilitam o estudo.



Lista com VETOR: V.S LDE circular - Questão 1

* Adicionar no início - O(n) - Linear ☹

Uma coisa interessante, que particularmente acho legal em vetores, é que temos uma lista de itens numerados sequencialmente por meio de índices.
O problema disso é que se nós desejarmos adicionar um elemento no início do vetor, vamos ter que "dar um shift à direita" e reindexar todos os 'n' itens do meu vetor.

* Adicionar no início - O(1) - constante. ☺

Já para adicionar no início da LDE circular a complexidade é O(1), pois tenho acesso direto ao ponteiro início, que representa o "início" do meu ciclo. Assim, após acessar esse ponteiro início é necessário adicionar o novo Nó e com cuidado realizar as operações necessárias para fechar o ciclo da minha LDE circular.

* Remover do início - O(n) - Linear ☹

Pela mesma lógica do adicionar, ao remover um elemento do início de um vetor, todos os elementos que vem depois dele devem sofrer um "shift para esquerda" para reindexar os itens. Isso acaba tornando a complexidade de tempo O(n) - Linear.

* Remover do início - O(1) - Constante ☺

Segue a mesma ideia do adicionar no início. Porém, nesse caso é necessário remover o Nó para onde o ponteiro início está apontando. Após isso, devemos tomar cuidado ao realizar as operações necessárias para fechar o "ciclo" da minha LDE circular. (implementação do remover do início, fim e meio estão presentes no arquivo questaob.c)