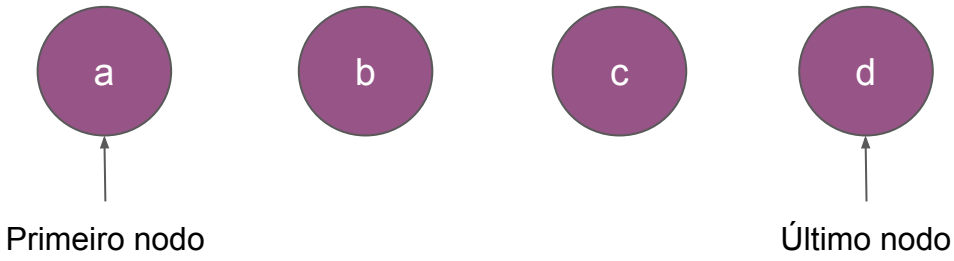
Listas Lineares

Listas Lineares

- Uma lista linear (ou simplesmente lista) é uma estrutura de dados formada por uma sequência de elementos do mesmo tipo armazenados na memória RAM
- Os elementos são chamados de **nodos** (*nodes*, em inglês) e podem conter dados de tipos primitivos ou estruturados
- O relacionamento entre os nodos é definido por sua posição em relação aos demais nodos da lista

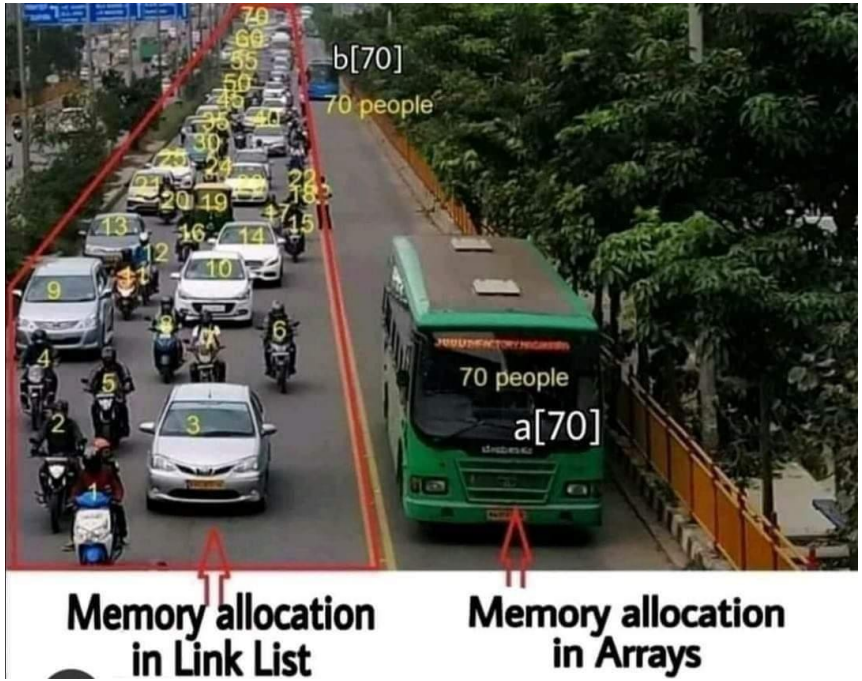


Listas Lineares

- Algumas operações sobre listas:
  - Criação da lista
  - Inserção de nodo
  - Exclusão de um nodo específico
  - Acesso a um nodo específico (para consulta ou alteração)
  - Contagem do número de nodos
  - Impressão da lista
  - Testar se está vazia
  - Esvaziamento da lista
  - Destruição da lista
  - …

Listas Lineares

- Uma lista linear pode ser implementada:
  - Por vetor
  - Por encadeamento

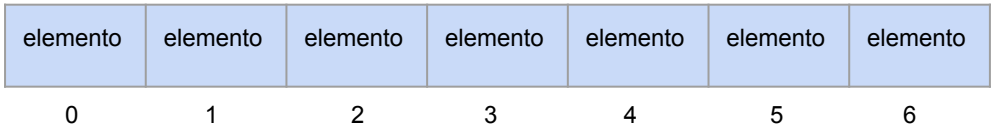


Lista com Vetor

- Representação por contiguidade física
- A ordem de cada nodo na lista é definida implicitamente pela posição ocupada por ele na memória
- Quais problemas ela apresenta?
  - Tamanho predefinido (possível fonte de desperdício de memória)
  - A inserção e a exclusão de nodos geralmente implicam na movimentação de um número considerável outros nodos para que a contiguidade física seja assegurada

```
typedef struct {
  int id;
  char nome[31];
  double salario;
} Funcionario;
```

```
Funcionario funcionarios[7];
```



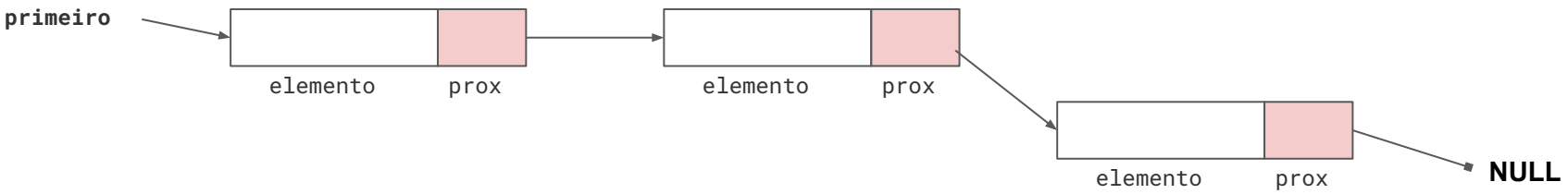
Lista Encadeada (*Linked List*)

- Utiliza alocação dinâmica
  - A cada inserção de um nodo, um novo espaço livre de memória é alocado
  - Os nodos são alocados em posições aleatórias da memória, e não de forma sequencial, como num vetor
    - Isso impossibilita o acesso direto aos elementos através do índice
- Como então saber a posição de cada nodo na lista? Como acessar um nodo específico? Como percorrer toda a lista?

Cada nodo armazena, além dos valores de seus campos, o endereço do próximo elemento

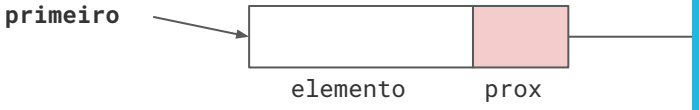
Lista Encadeada

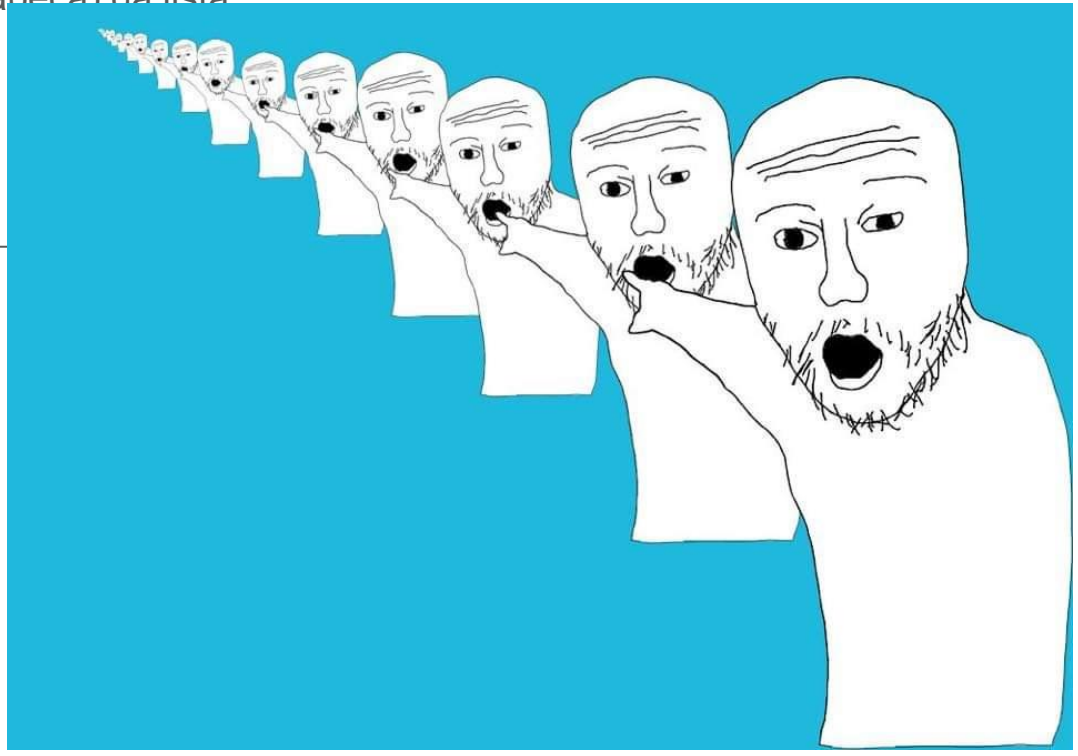
- Precisamos de um ponteiro que aponte para o início da lista (primeiro nodo)
  - Também chamado de *head* (cabeça) da lista
- O último aponta para NULL



Lista Encadeada

- Precisamos de um ponteiro que aponte para o início da lista (primeiro nodo)
  - Também chamado de *head* (cabeça) da lista
- O último aponta para NULL

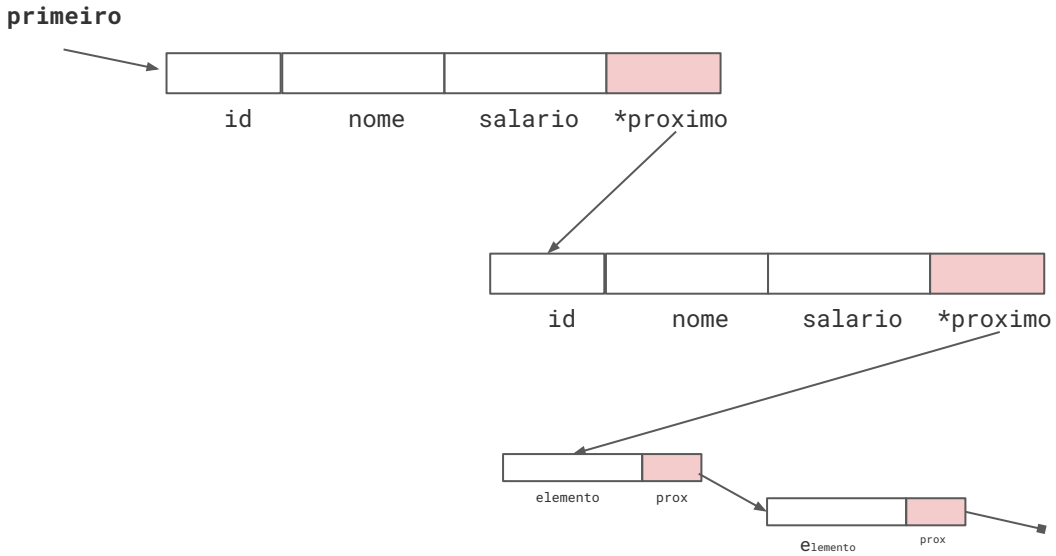




Lista Encadeada

- Exemplo:

```c
struct funcionario{
  int id;
  char nome[31];
  double salario;
  struct funcionario *proximo;
};
typedef struct funcionario Funcionario;
```



Lista Encadeada

- Criação da lista vazia
  - Operação que deve ser realizada antes de todas as demais

```
Funcionario *primeiro; //head, first... como quiser
primeiro = NULL;
```

Lista Encadeada

- Criação da lista com inserção do primeiro nodo

```c
struct funcionario{
  int id;
  char nome[31];
  double salario;
  struct funcionario *proximo;
}
typedef struct funcionario Funcionario;
```

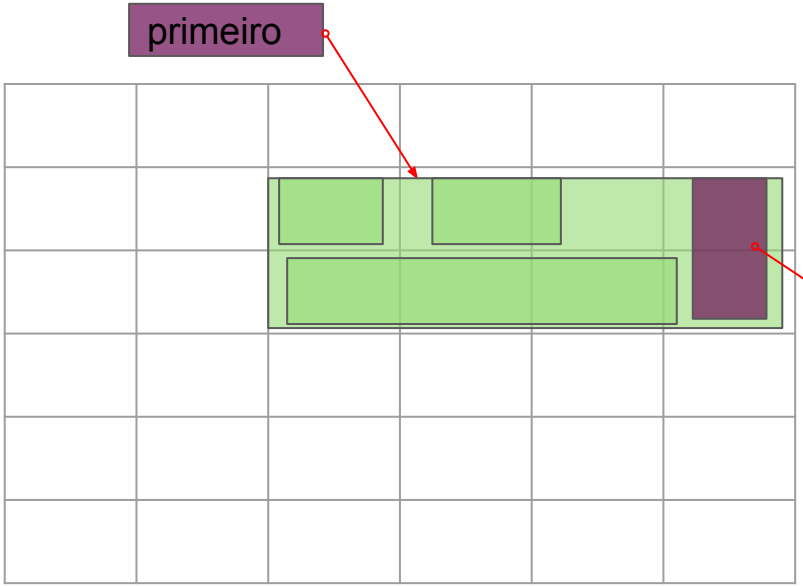
```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Memória

primeiro

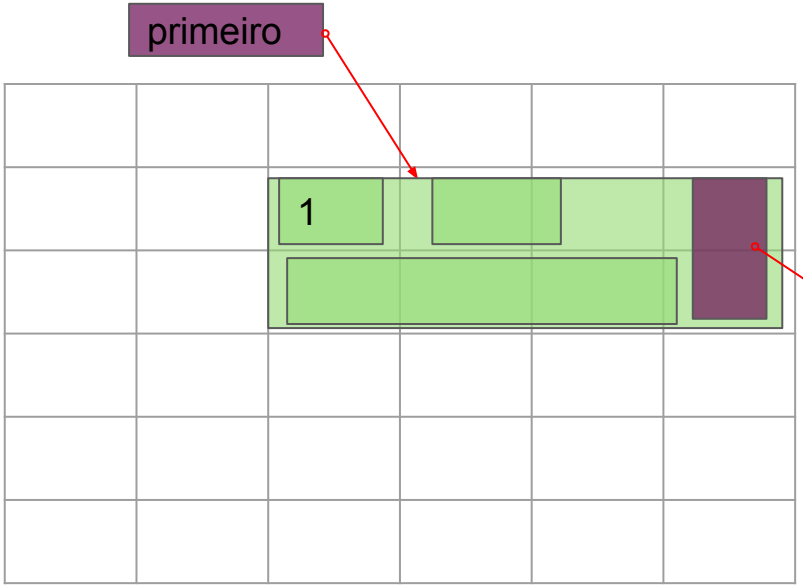

```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Memória



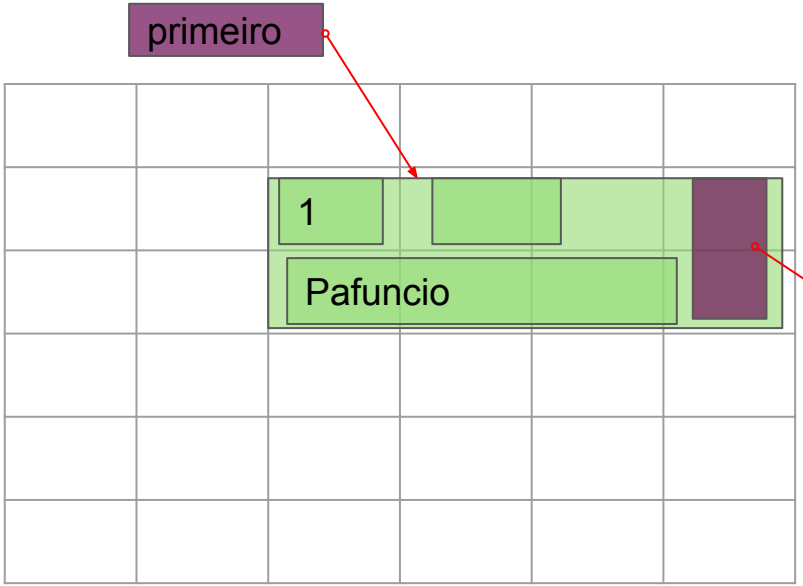

```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Memória



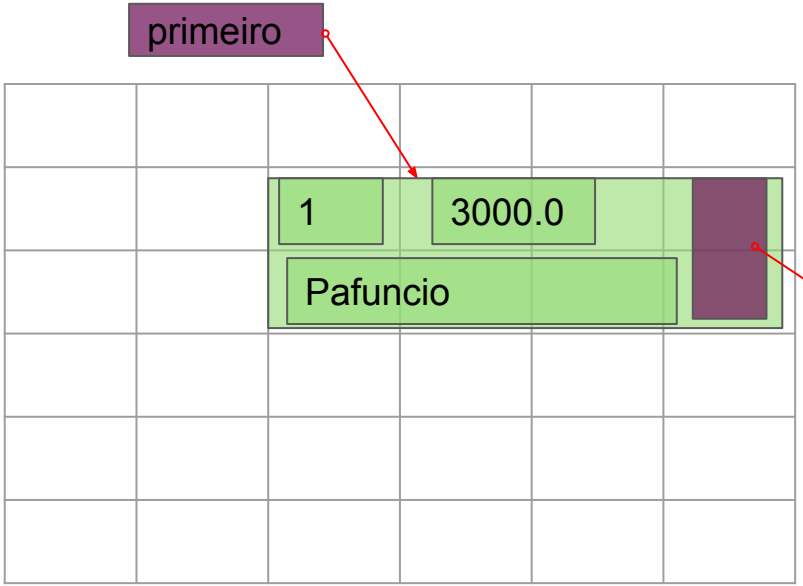

```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Memória



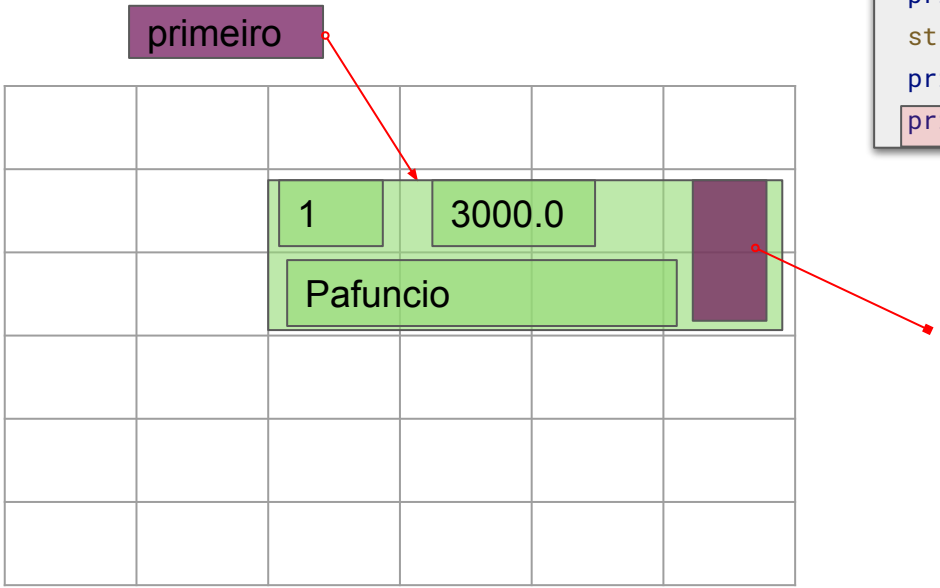

```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Memória




```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Memória




```c
Funcionario *primeiro; //head, first... como quiser

primeiro = malloc(sizeof(Funcionario));

primeiro->id = 1;
strcpy(primeiro->nome, "Pafuncio");
primeiro->salario = 3000.0;
primeiro->proximo = NULL;
```

Lista Encadeada

- Inserção de outros nodos
  - É necessário pensar no encadeamento da lista

A inserção pode ocorrer:

- No início da lista
- No meio da lista
- No final da lista

Veremos esta como exemplo

Lista Encadeada

- Inserção de outros nodos
  - É necessário pensar no encadeamento da lista

```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Lista Encadeada

- Inserção de outros nodos
  - É necessário pensar no encadeamento da lista

```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Criando um auxiliar e alocando memória

Lista Encadeada

- Inserção de outros nodos
  - É necessário pensar no encadeamento da lista

```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Preenchendo os valores

Lista Encadeada

- Inserção de outros nodos
  - É necessário pensar no encadeamento da lista

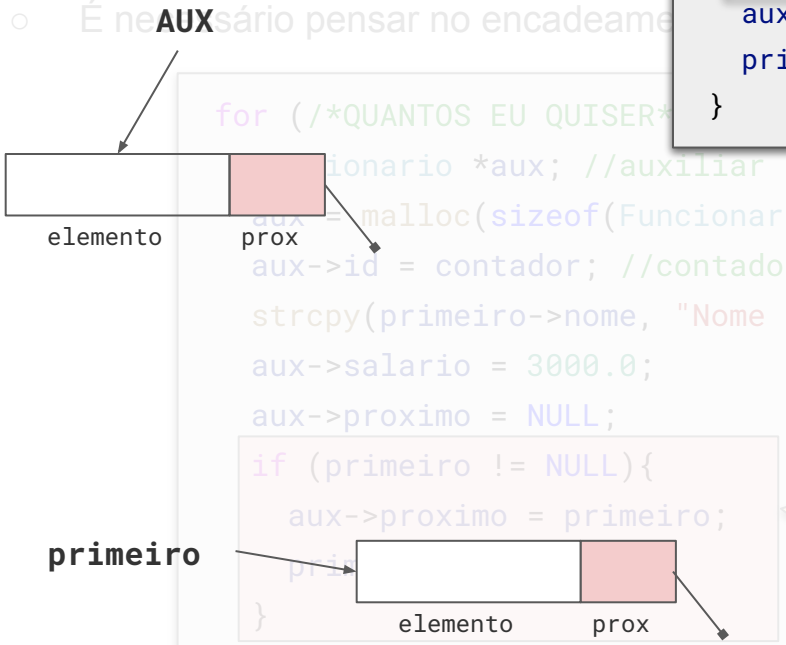
```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Aqui está ocorrendo o encadeamento da lista. Perceba que se o primeiro existe, ele vira o próximo do auxiliar, e o auxiliar se torna o primeiro.

Lista Encadeada

- Inserção de outros nodos
  - É necessário pensar no encadeamento

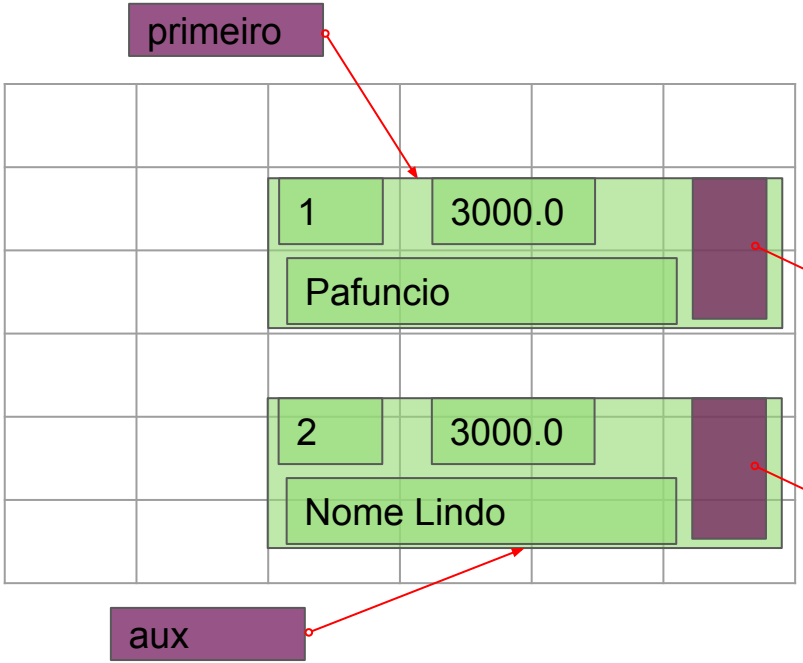
```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```



Aqui está ocorrendo o encadeamento da lista. Perceba que se o primeiro existe, ele vira o próximo do auxiliar, e o auxiliar se torna o primeiro.

Memória

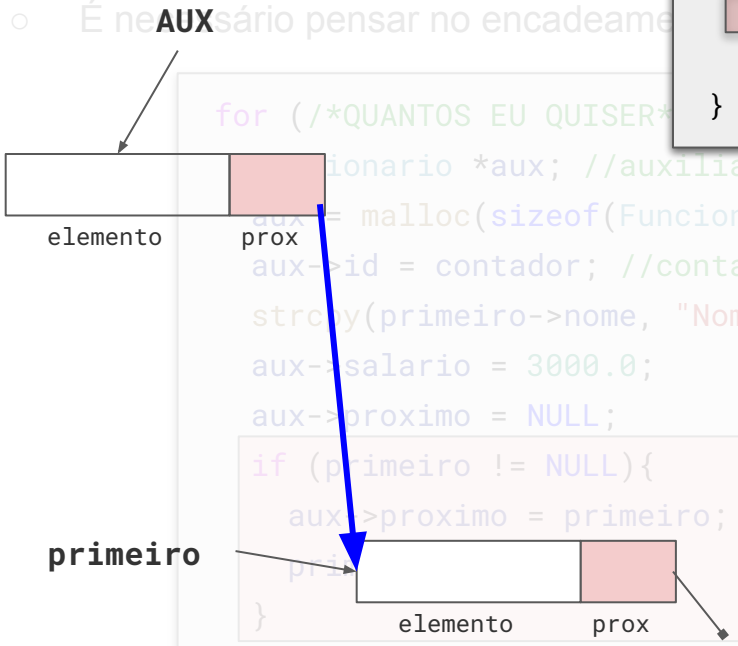
```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```



Lista Encadeada

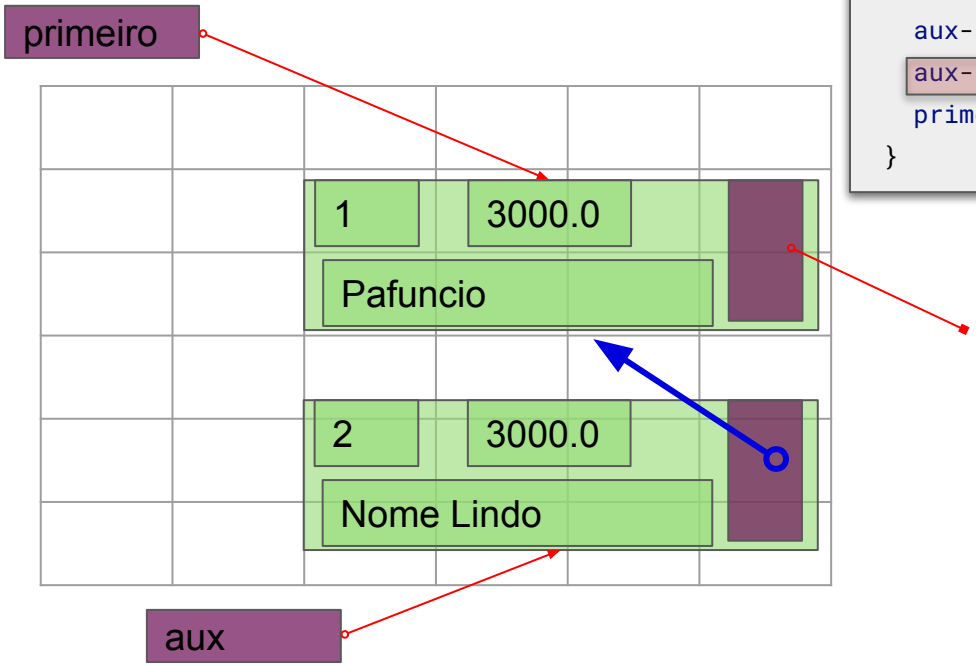
- Inserção de outros nodos
  - É necessário pensar no encadeamento

```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```



Aqui está ocorrendo o encadeamento da lista. Perceba que se o primeiro existe, ele vira o próximo do auxiliar, e o auxiliar se torna o primeiro.

Memória

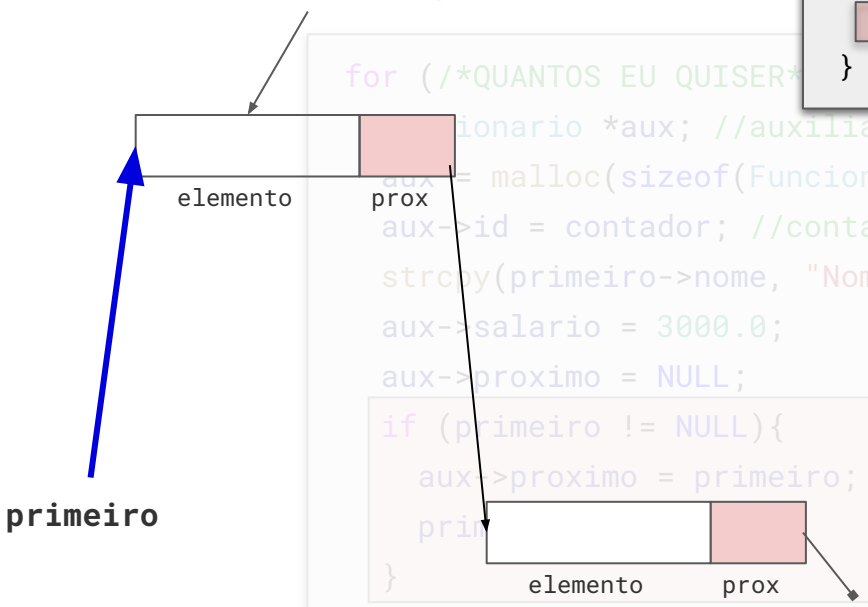



```c
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Lista Encadeada

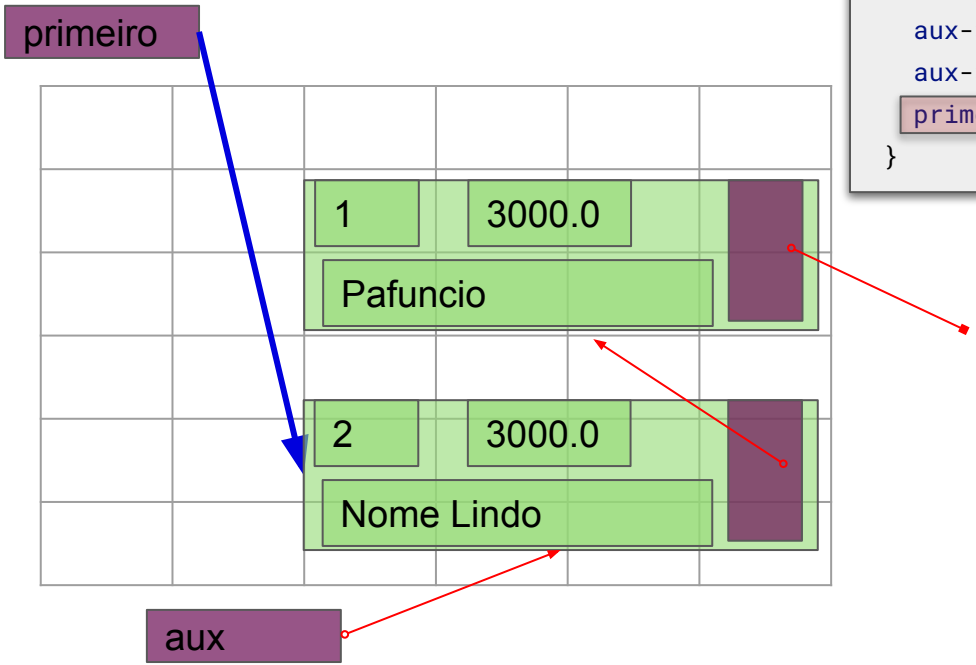
- Inserção de outros nodos
  - É necessário pensar no encadeamento

```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```



Aqui está ocorrendo o encadeamento da lista. Perceba que se o primeiro existe, ele vira o próximo do auxiliar, e o auxiliar se torna o primeiro.

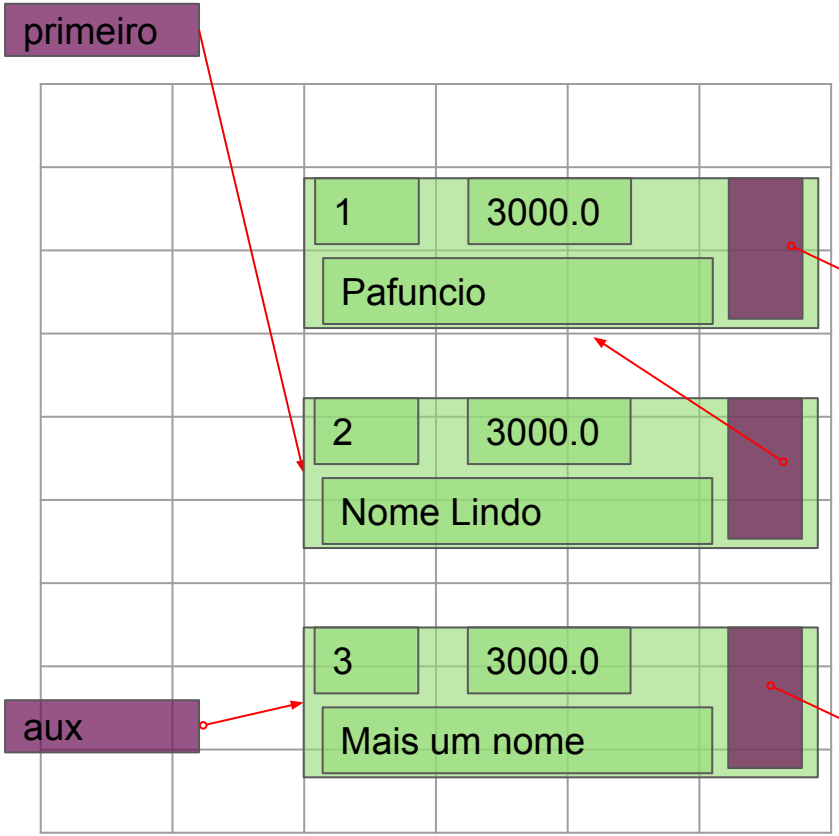
Memória



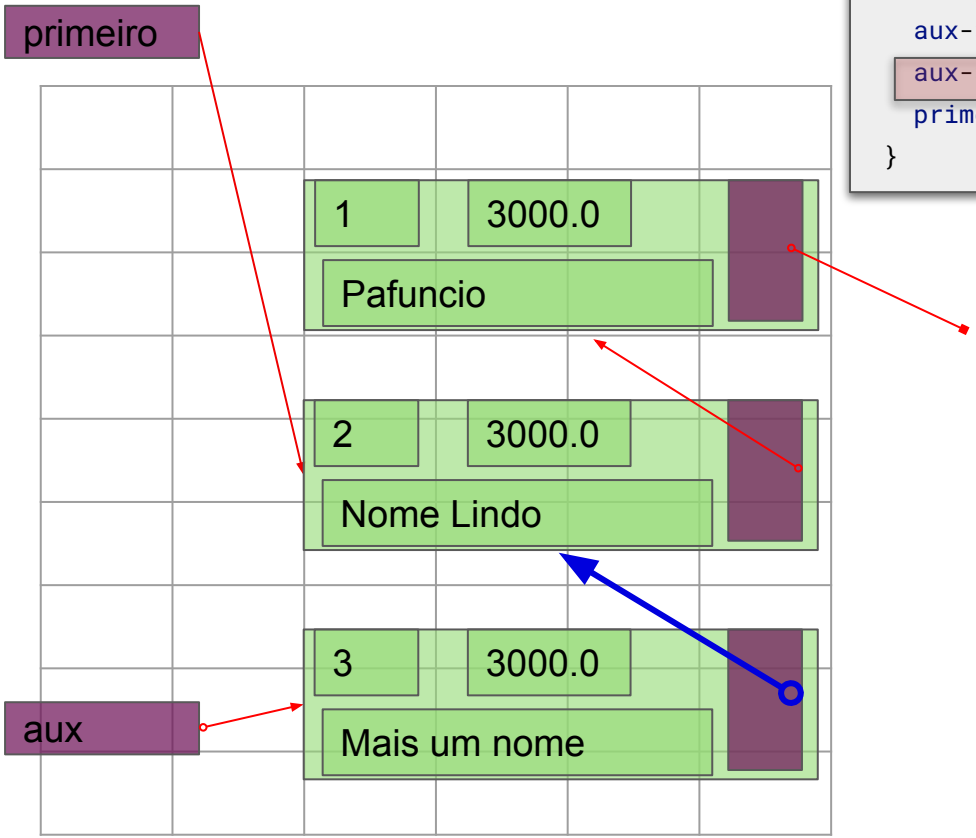

```c
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Memória

```
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

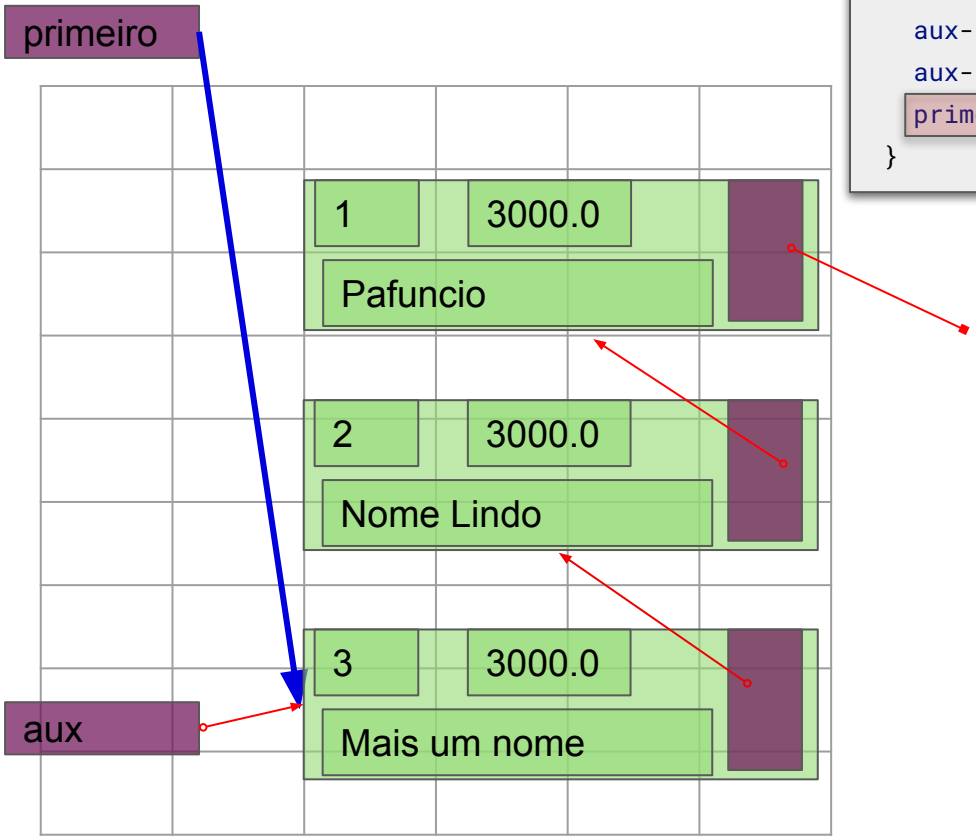


Memória




```c
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Memória




```c
for (/*QUANTOS EU QUISER*/){
  Funcionario *aux; //auxiliar
  aux = malloc(sizeof(Funcionario));
  aux->id = contador; //contador é uma variável qualquer
  strcpy(aux->nome, "Nome Lindo");
  aux->salario = 3000.0;
  aux->proximo = primeiro;
  primeiro = aux;
}
```

Lista Encadeada

- Impressão da lista
  - Deve-se iterar sobre todos os elementos, partindo do primeiro

Qualquer percurso utilizará esta mesma lógica:
- Percorrer a lista até o fim
- Percorrer até encontrar um elemento específico (adicionamos uma condição e um break)

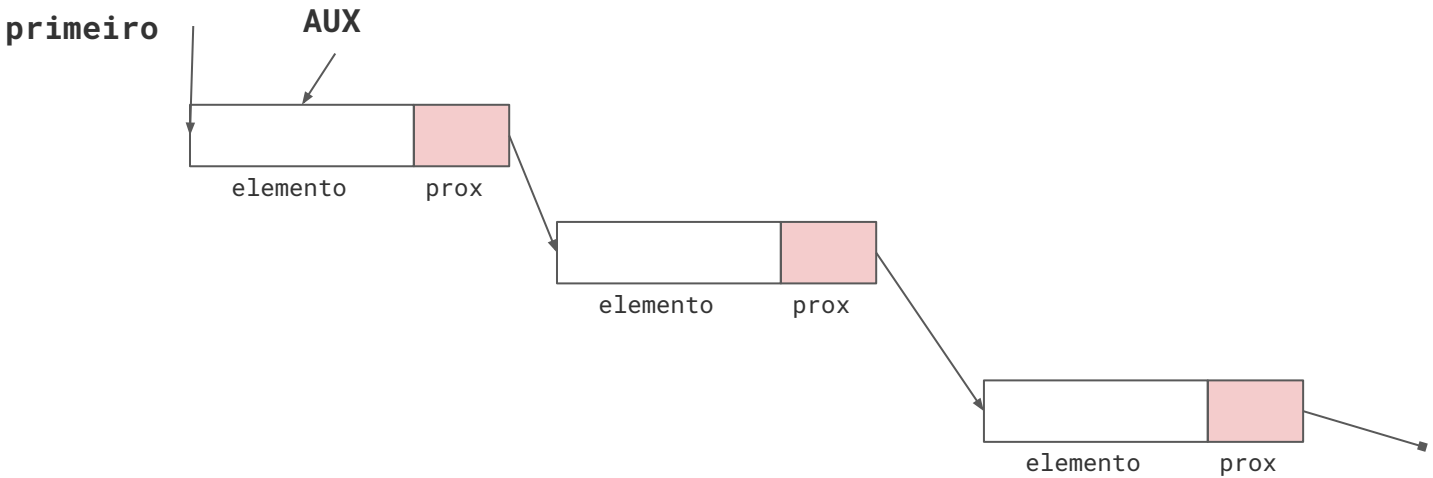
```
Funcionario *aux; //vai ser nosso 'contador'
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  //aqui aux vale o elemento atual na lista.
  printf("Funcionario id: %d, nome: %s, salario: %f\n",  aux->id, aux->nome, aux->salario);
}
```

Lista Encadeada

- Imprimindo os elementos
  -

```c
Funcionario *aux; //va
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  //aqui aux vale o elemento atual na lista.
  printf("Funcionario id: %d, nome: %s, salario: %f\n",  aux->id, aux->nome, aux->salario);
}
```

Iniciamos pelo primeiro elemento

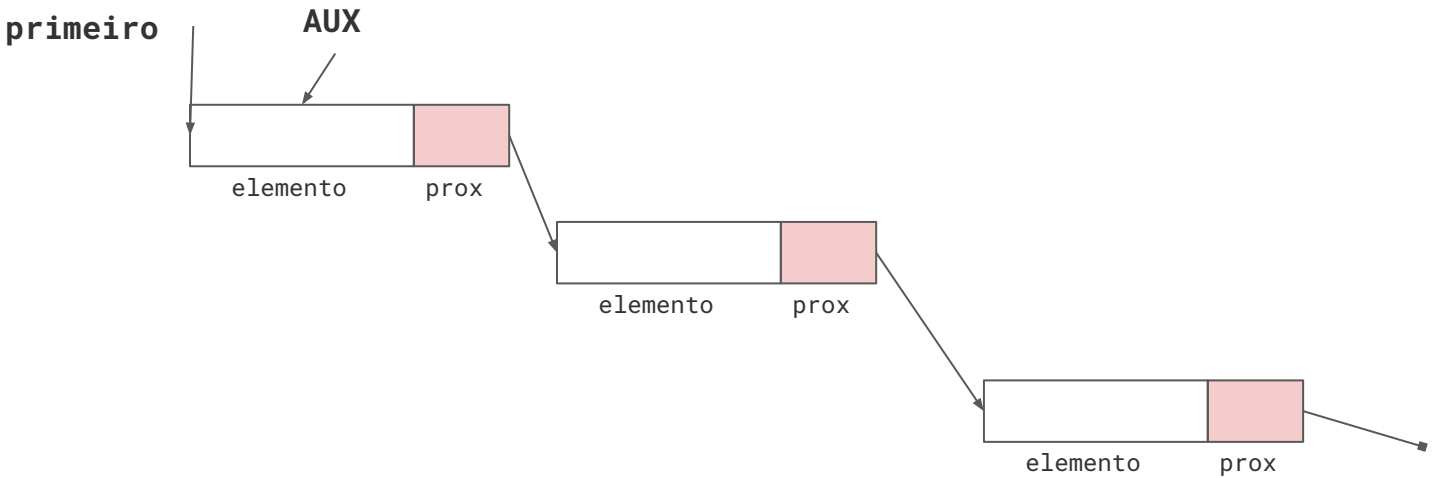


Lista Encadeada

- Imprimindo os elementos
  -

```
Funcionario *aux; //vai ser nosso 'contador'
for (aux = primeiro; a                    ){
  //aqui aux vale o elemen        ual na lista.
  printf("Funcionario id: %d, nome: %s, salario: %f\n",  aux->id, aux->nome, aux->salario);
}
```

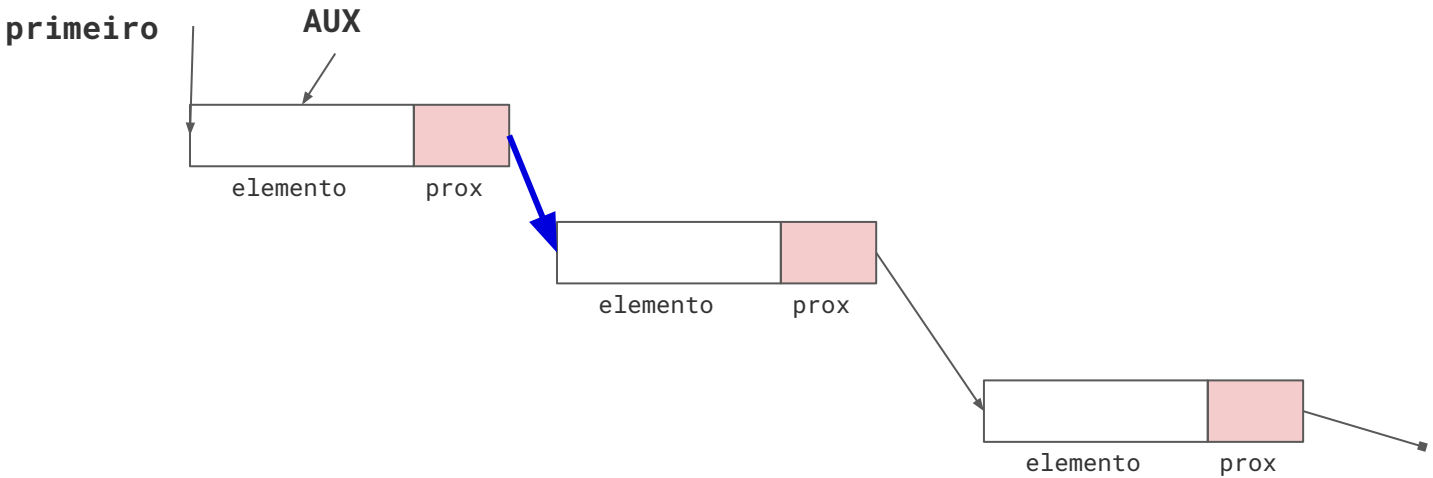
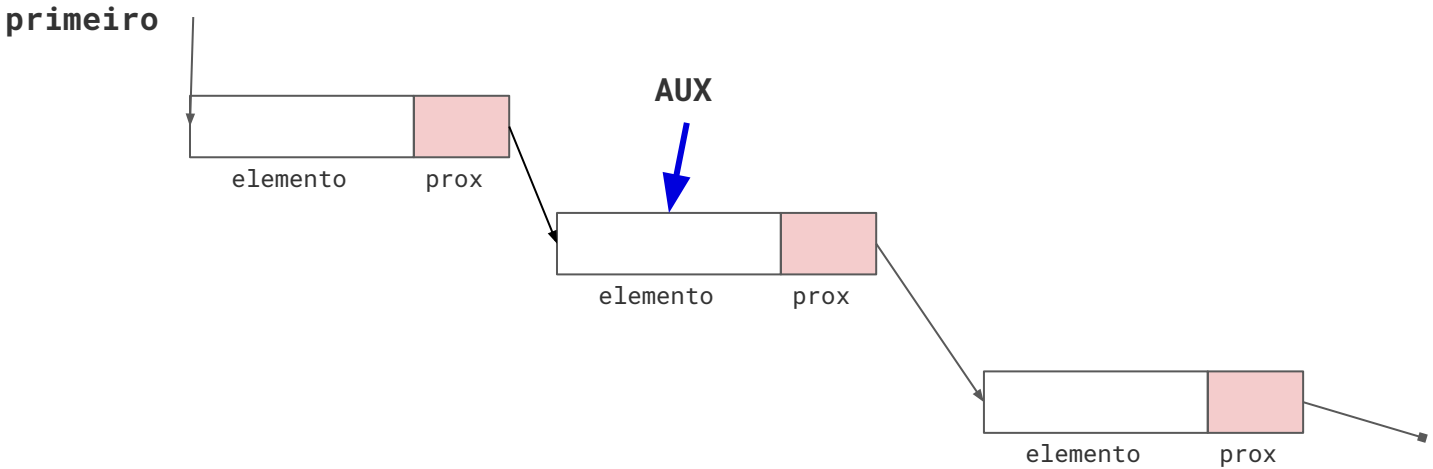
Printa o AUX



Lista Encadeada

- Imprimindo os elementos
  - ```
    Funcionario *aux; //vai ser nosso 'contador'
    for (aux = primeiro; aux != NULL; aux = aux->proximo){
      //aqui aux vale o elemento atual
      printf("Funcionario id: %d, nome:                    aux->nome, aux->salario);
    }
    ```

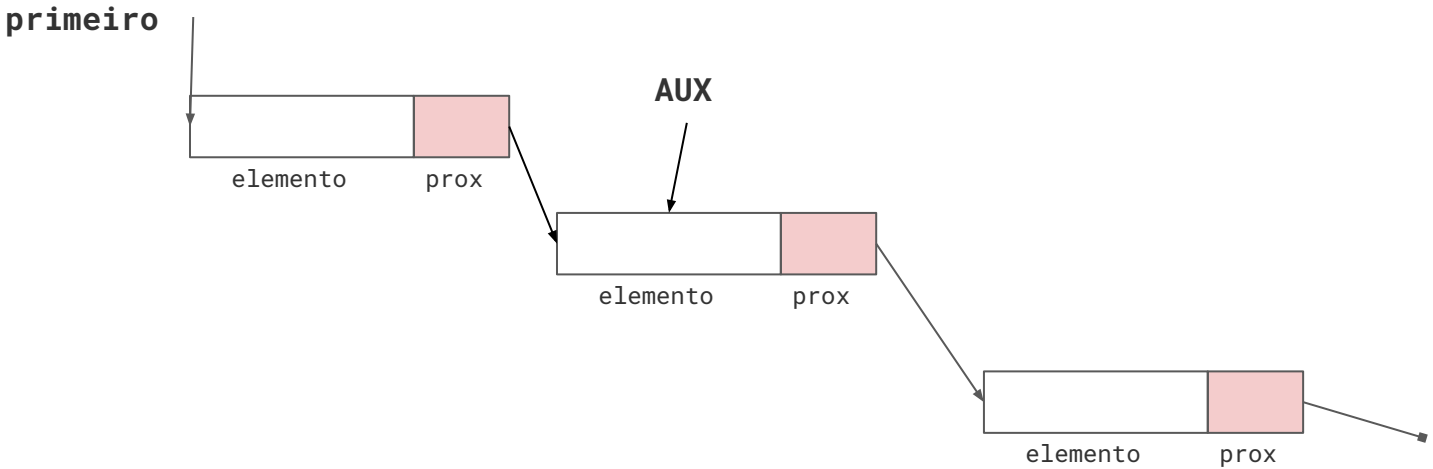
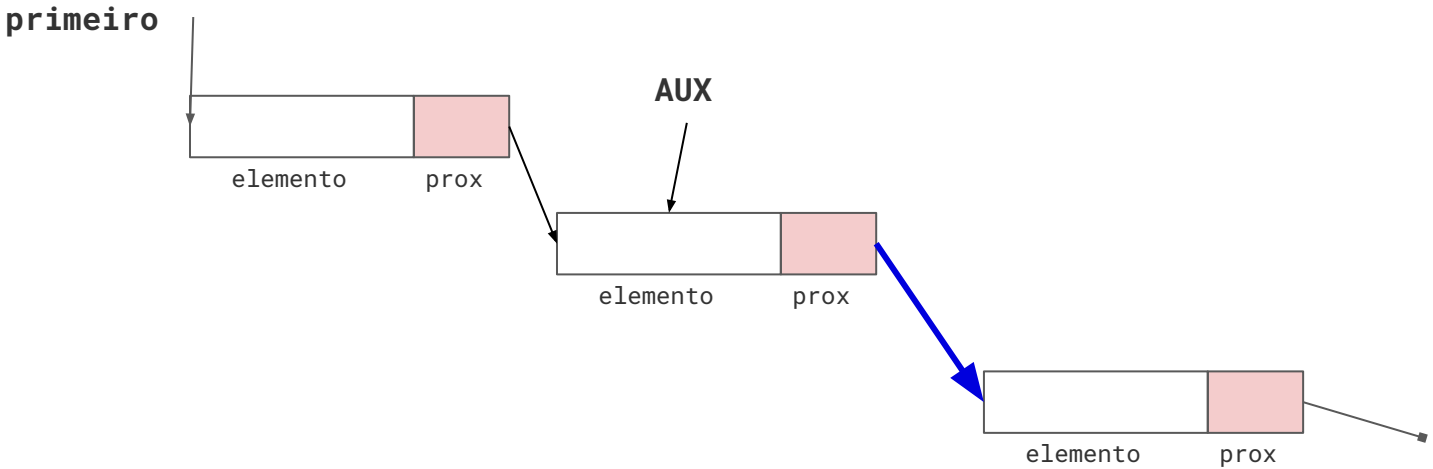
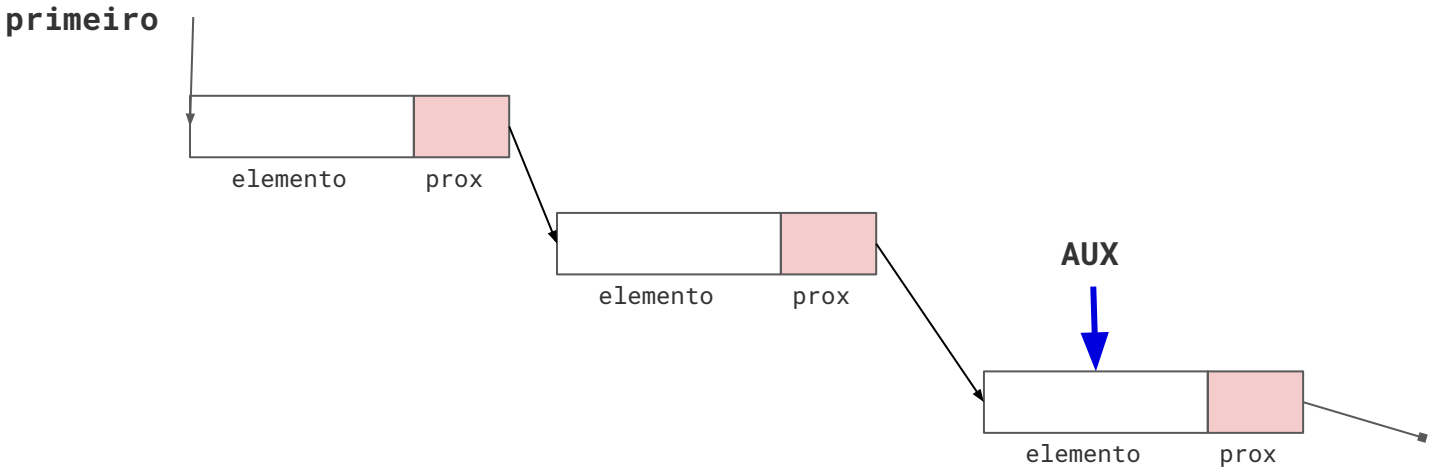
Faz o aux apontar para o próximo


    ```

Lista Encadeada

- Imprimindo os elementos
  - ```
    Funcionario *aux; //vai ser nosso 'contador'
    for (aux = primeiro; aux != NULL; aux = aux->proximo){
      //aqui aux vale o elemento atual
      printf("Funcionario id: %d, nome:                    aux->nome, aux->salario);
    }
    ```

Faz o aux apontar para o próximo


    ```

Lista Encadeada

- Imprimindo os elementos
  - ```
    Funcionario *aux; //vai ser nosso 'contador'
    for (aux = primeiro; a                    ){
      //aqui aux vale o elemen      ual na lista.
      printf("Funcionario id: %d, nome: %s, salario: %f\n",  aux->id, aux->nome, aux->salario);
    }
    ```

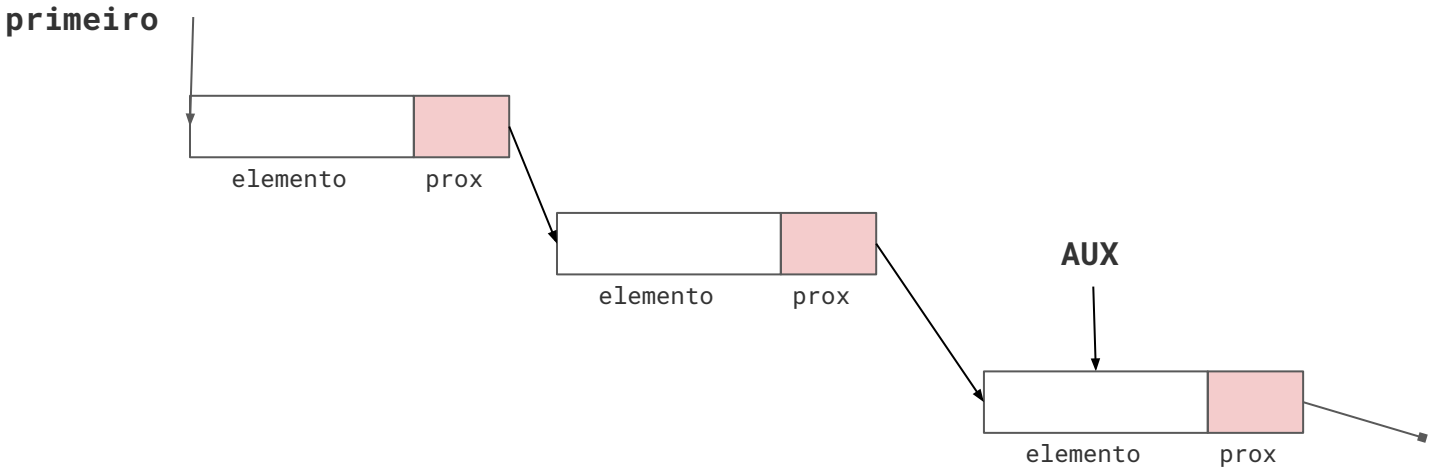
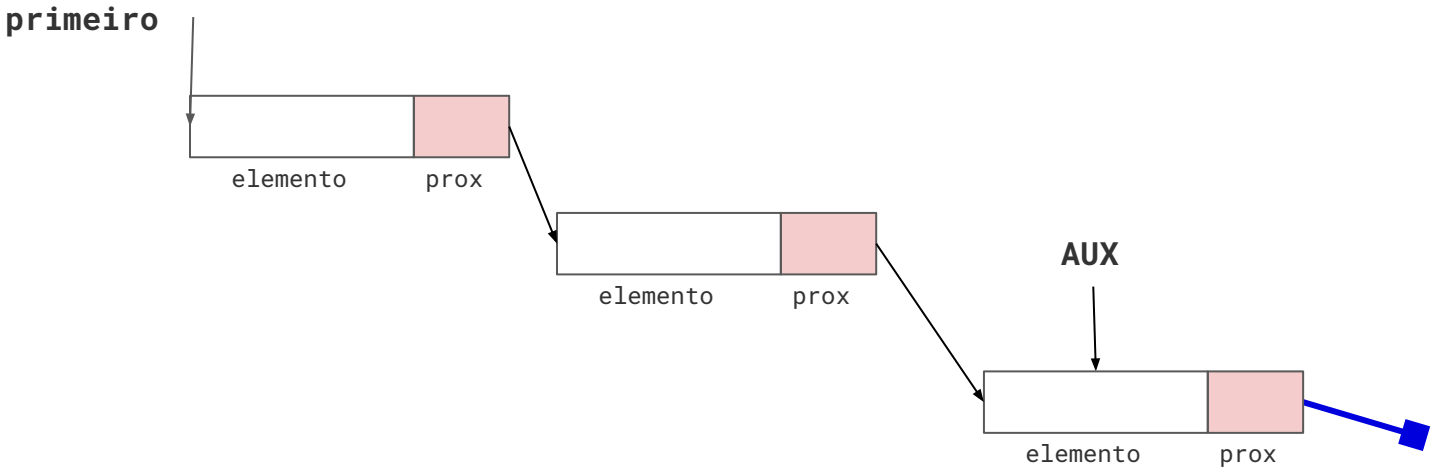
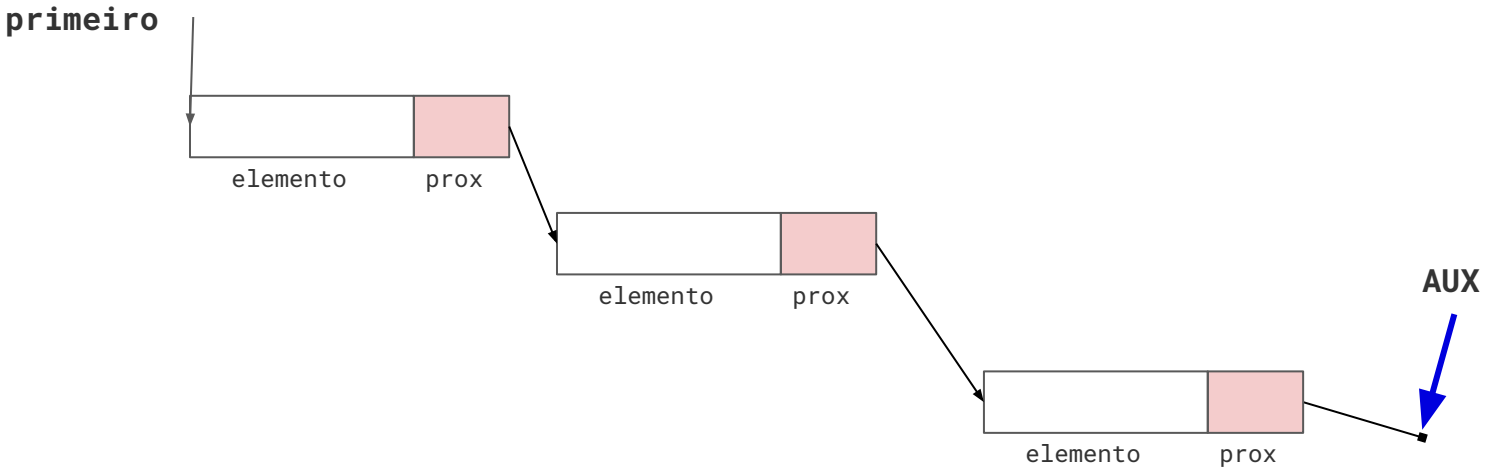
Printa o AUX


    ```

Lista Encadeada

- Imprimindo os elementos
  -

```
Funcionario *aux; //vai ser nosso 'contador'
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  //aqui aux vale o elemento atual n
  printf("Funcionario id: %d, nome:                    aux->nome, aux->salario);
}
```

Faz o aux apontar para o próximo



Lista Encadeada

- Imprimindo os elementos
  - ```
    Funcionario *aux; //vai ser nosso 'contador'
    for (aux = primeiro; aux != NULL; aux = aux->proximo){
      //aqui aux vale o elemento atual
      printf("Funcionario id: %d, nome:                    aux->nome, aux->salario);
    }
    ```

Faz o aux apontar para o próximo


    ```

Lista Encadeada

- Imprimindo os elementos
  - ```
    Funcionario *aux; //vai ser nosso 'contador'
    for (aux = primeiro; a                    ){
      //aqui aux vale o elemen        ual na lista.
      printf("Funcionario id: %d, nome: %s, salario: %f\n",  aux->id, aux->nome, aux->salario);
    }
    ```

Printa o AUX

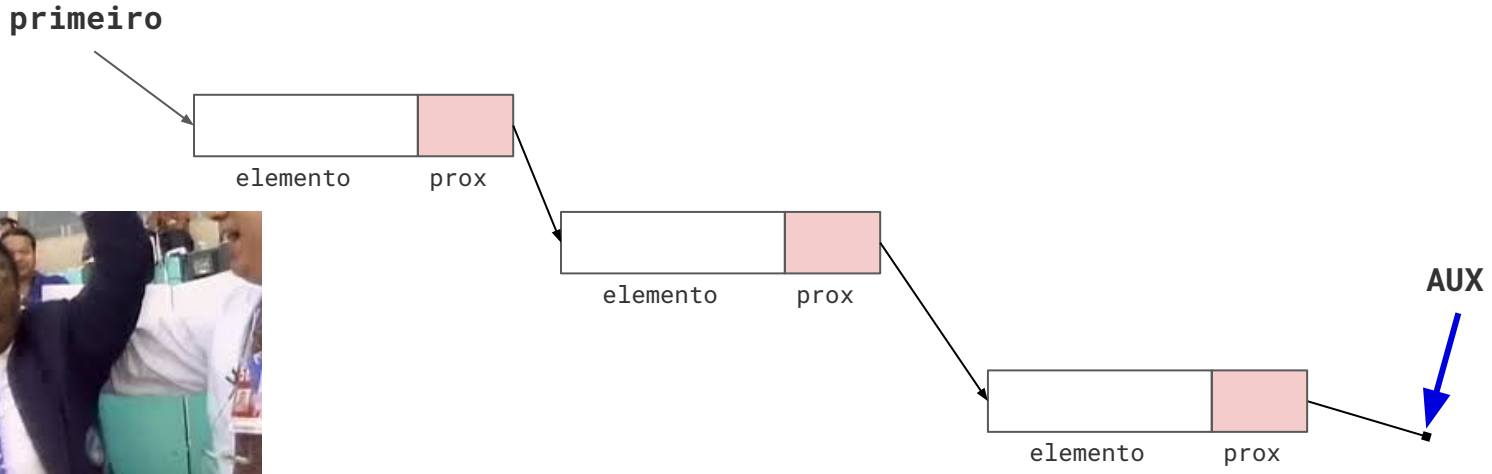

    ```

Lista Encadeada

- Imprimindo os elementos
  -

```
Funcionario *aux; //vai ser nosso 'contador'
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  //aqui aux vale o elemento atual n
  printf("Funcionario id: %d, nome:                    aux->nome, aux->salario);
}
```

Faz o aux apontar para o próximo



Lista Encadeada

- Imprimindo os elementos
  - ```
    Funcionario *aux; //vai ser nosso 'contador'
    for (aux = primeiro; aux != NULL; aux = aux->proximo){
      //aqui aux vale o elemento atual
      printf("Funcionario id: %d, nome:                    aux->nome, aux->salario);
    }
    ```

Faz o aux apontar para o próximo


    ```

Lista Encadeada

- Imprimindo os elementos
  -

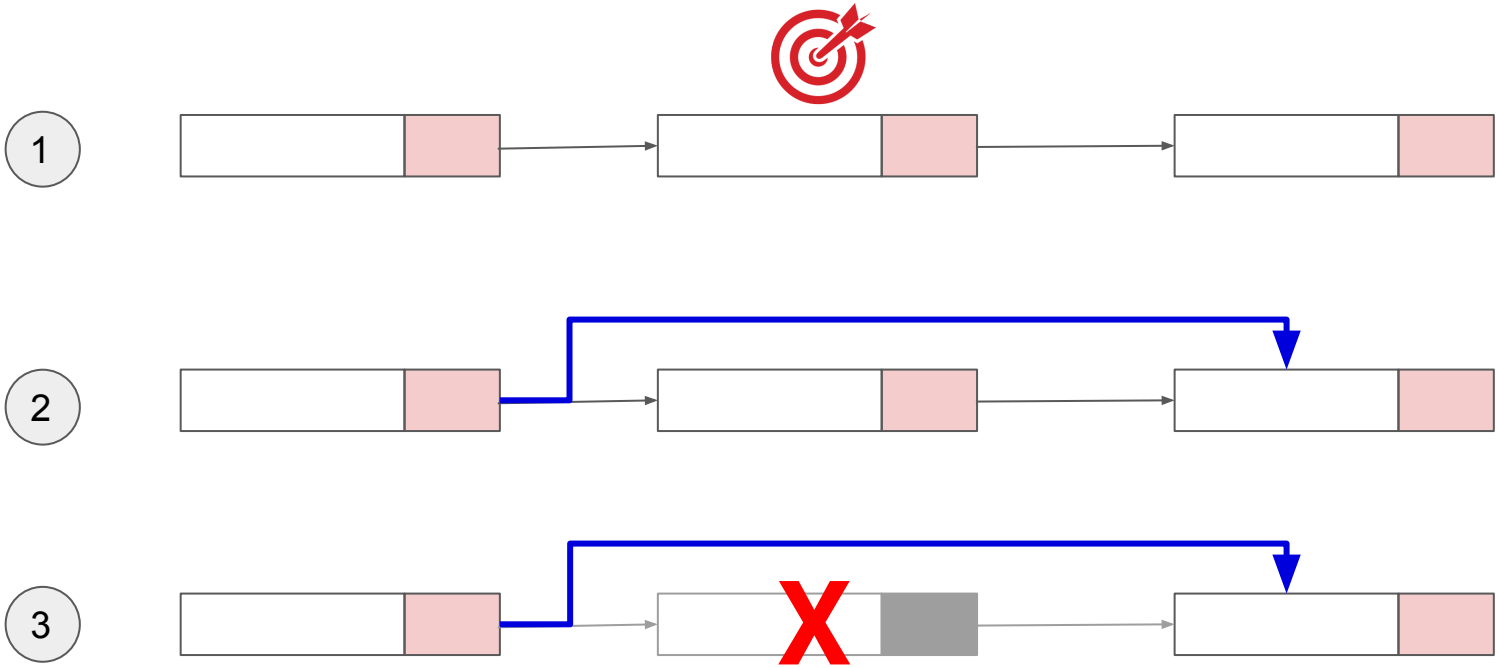
```
Funcionario *aux; //vai ser nosso 'contador'
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  //aux vale o elemento atu
  printf("Funcionario id: %d, nc                    id, aux->nome, aux->salario);
}
```

Agora, **aux** é NULL.



Lista Encadeada

- Exclusão de um nodo



Lista Encadeada

- Exclusão de um nodo
  - Percorrer a lista até encontrar o nodo a ser removido

```
Funcionario *aux, *anterior;
int idDelete = 1; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){

  }
  anterior = aux; //controla o anterior
}
```

encontrou

Lista Encadeada

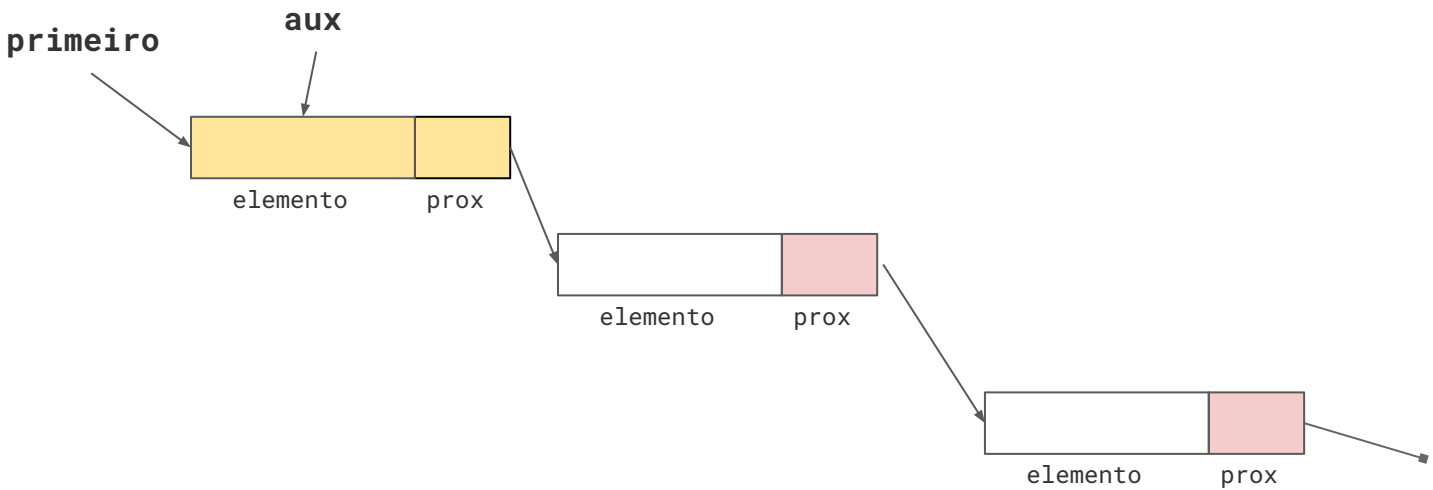
- Exclusão de um nodo
  - Percorrer a lista até encontrar o nodo a ser removido
  - Atualizar o encadeamento
    - É o primeiro? Refazer o ponteiro do primeiro (apontar para quem até então era o segundo)
    - Não é o primeiro? O anterior do excluído passará a apontar para o seguinte a ele
  - Apagar o nodo

```
Funcionario *aux, *anterior;
int idDelete = 1; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```

Lista Encadeada

Exclusão do primeiro nodo

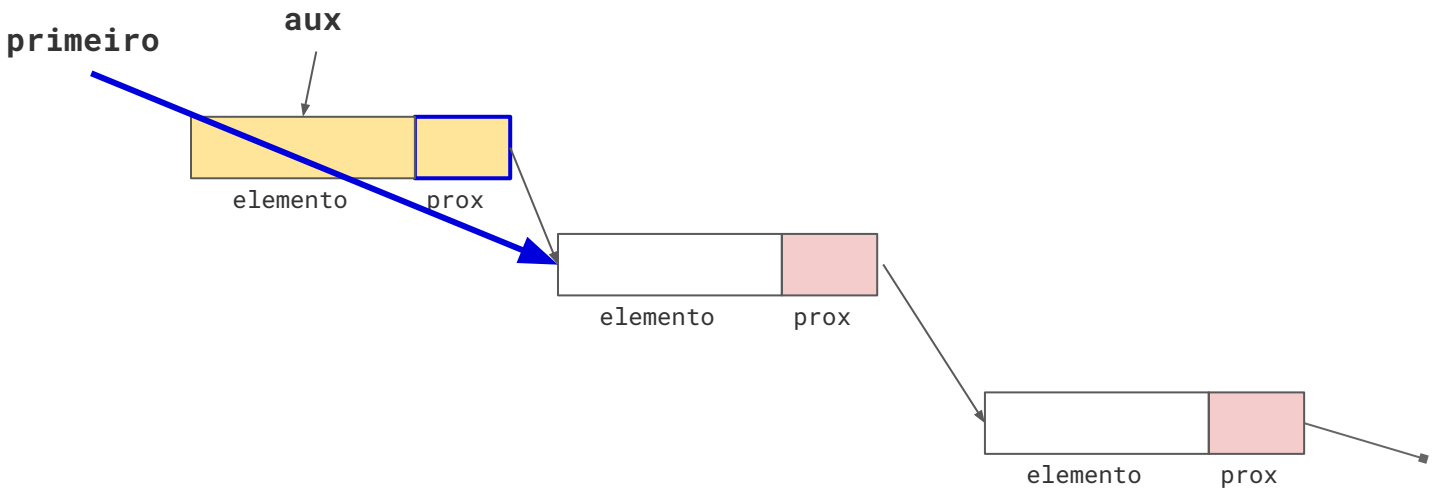
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

Exclusão do primeiro nodo

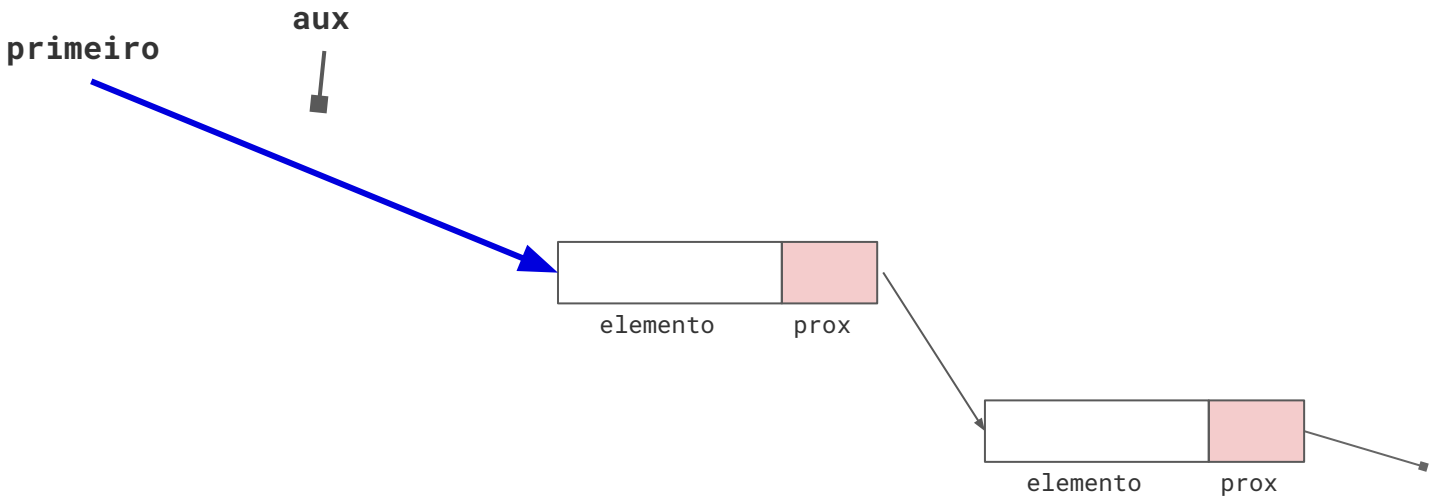
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

Exclusão do primeiro nodo

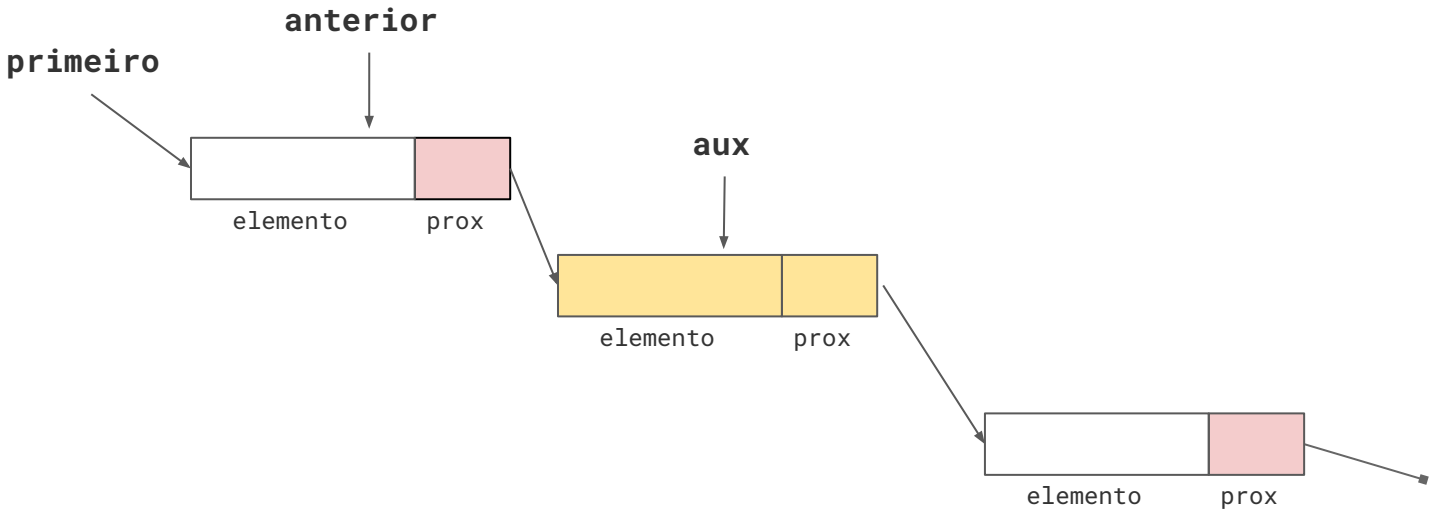
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

Exclusão de nodo após o primeiro

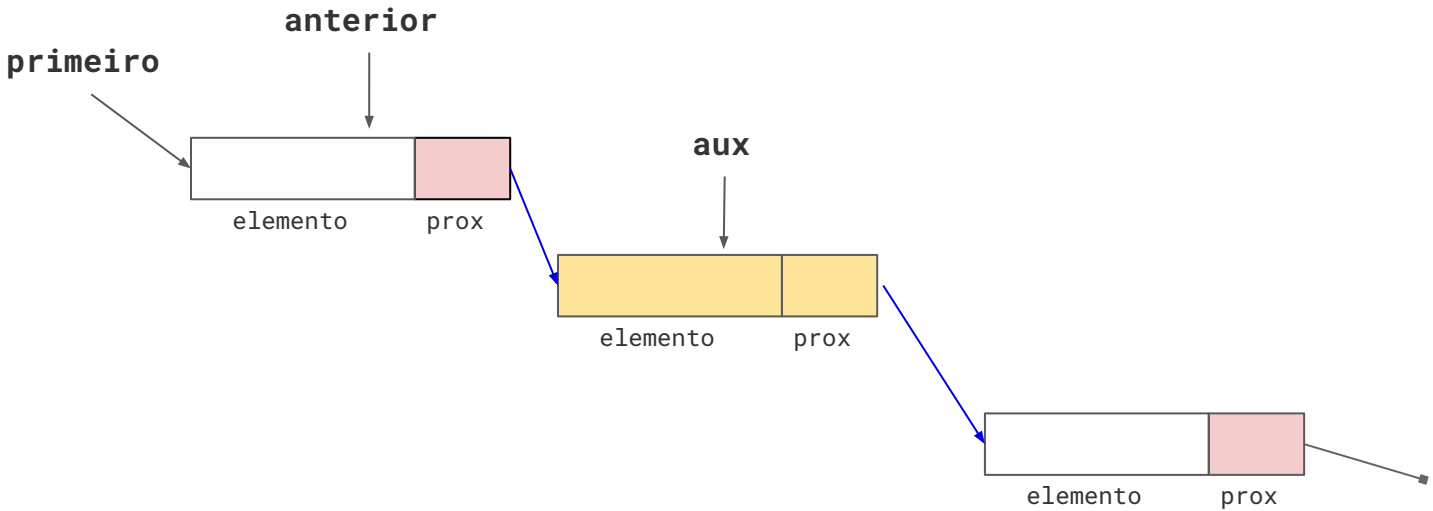
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

Exclusão de nodo após o primeiro

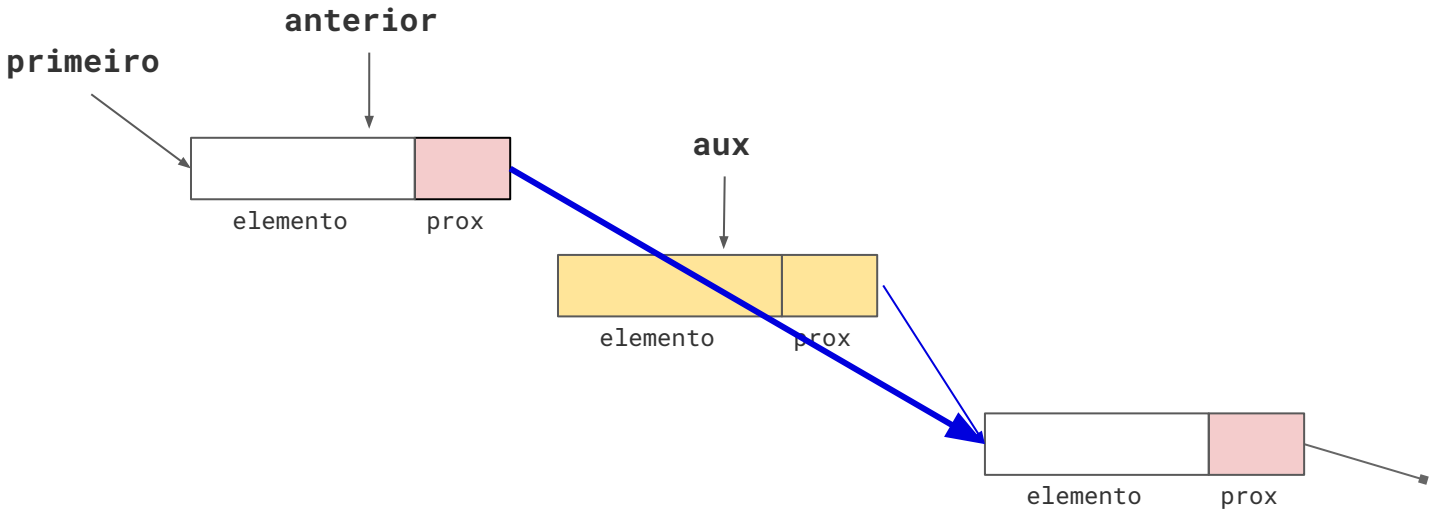
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

Exclusão de nodo após o primeiro

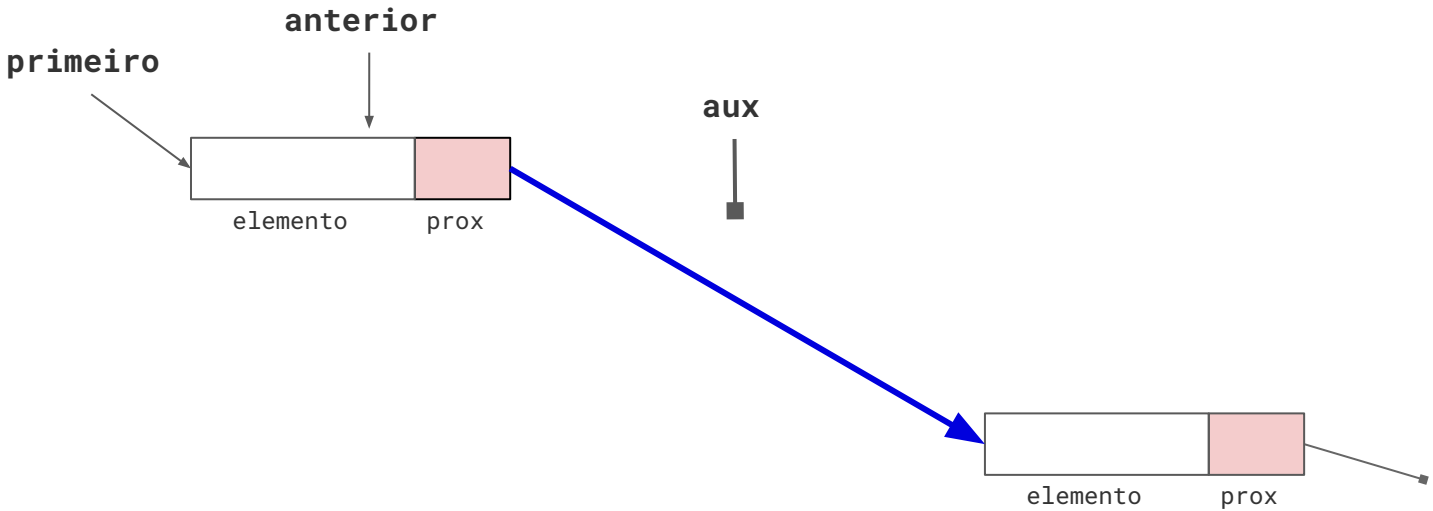
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

Exclusão de nodo após o primeiro

```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta para o segundo.
    } else {
      anterior->proximo = aux->proximo; //anterior aponta para o proximo de aux;
    }
    free(aux); //apaga o aux
    break;
  }
  anterior = aux; //controla o anterior
}
```



Lista Encadeada

- Liberando a lista
  - Liberar todos os elementos

```
while (primeiro != NULL){
  aux = primeiro;
  primeiro = primeiro->proximo;
  free(aux);
}
```

Lista Encadeada

- Estrutura da lista
  - Existem várias formas:

```c
struct funcionario{
  int id;
  char nome[TAM_NOME+1];
  double salario;
  struct funcionario *proximo;
};
typedef struct funcionario Funcionario;
```

```c
typedef struct{
  Funcionario *primeiro;
} Lista;
```

Lista Encadeada

- Estrutura da lista
  - Existem várias formas:

```c
typedef struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} Funcionario;
```

```c
struct lista {
  Funcionario *elemento;
  struct lista *proximo;
};
typedef struct lista Lista;
```

Lista Encadeada

- Estrutura da lista
  - Existem várias formas:

```c
typedef struct {
  int id;
  char nome[TAM_NOME+1];
  double salario;
} Funcionario;
```

```c
struct lista {
  Funcionario *elemento;
  struct lista *proximo;
};
typedef struct lista Lista;
```

```c
Lista *primeiro;
```

Exercício

1. Neste exercício, você deve implementar operações sobre uma lista com nodos do tipo `Funcionario` definido neste slide. Você deve implementar o seguinte:
   a. Uma função para inserir um nodo no início da lista;
   b. Uma função para inserir um nodo ao final da lista;
   c. Uma função que recebe o id do funcionário e o remove da lista;
   d. Uma função que imprime toda a lista;
   e. Uma função recursiva que imprime os valores dos nodos de trás para frente;
   f. Uma função que conta o número de nodos da lista;
   g. Uma função que destrói a lista (libera a memória da lista).