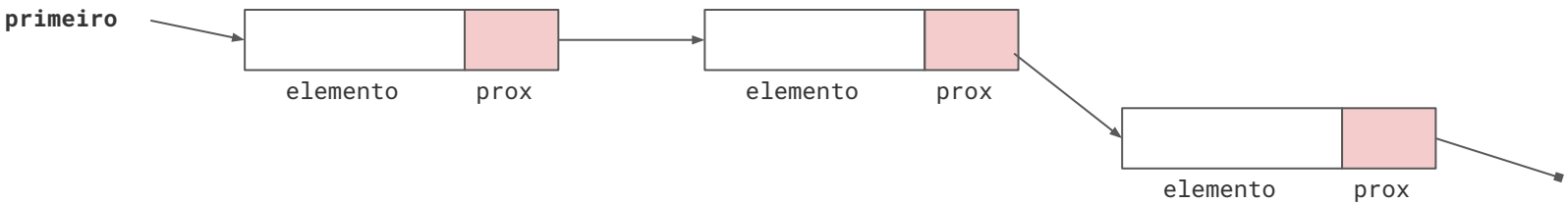
Listas

Prof. Andrei Braga
Prof. Geomar Schreiner

Lista encadeada

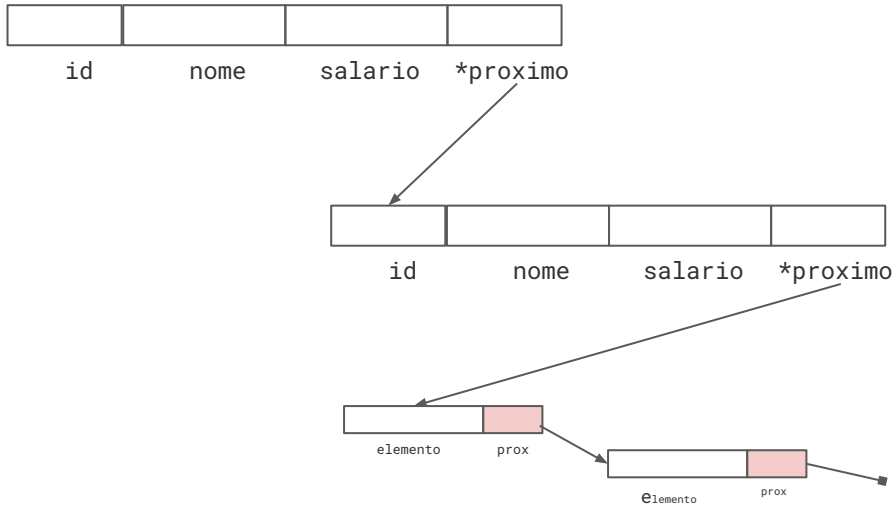
- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
  - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista



Lista encadeada

- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
  - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista

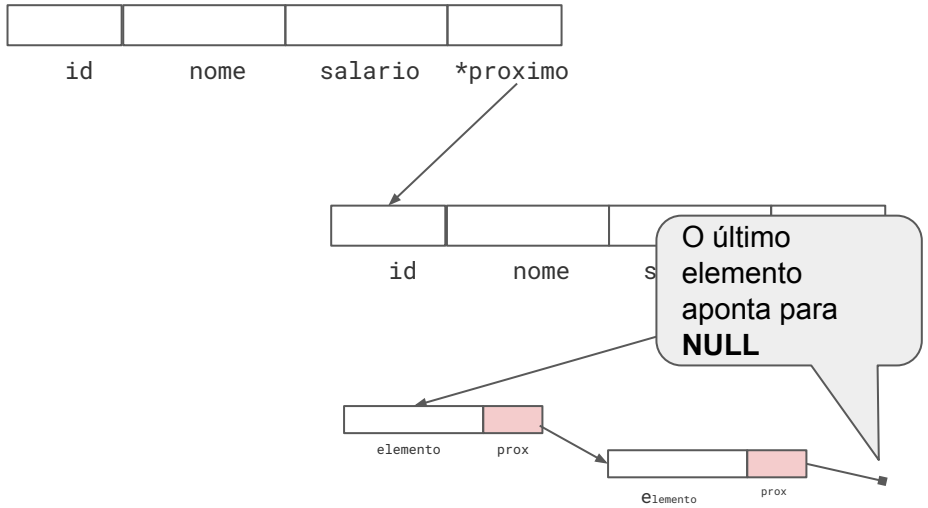
```c
struct funcionario{
  int id;
  char nome[TAM_NOME+1];
  double salario;
  struct funcionario *proximo;
};
typedef struct funcionario Funcionario;
```



Lista encadeada

- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
  - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista

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



Lista encadeada

- Deletar um elemento da lista
  - Precisa percorrer a lista até encontrar
  - Depois de encontrar precisa refazer o aponteiramento
  - Depois apagar o elemento

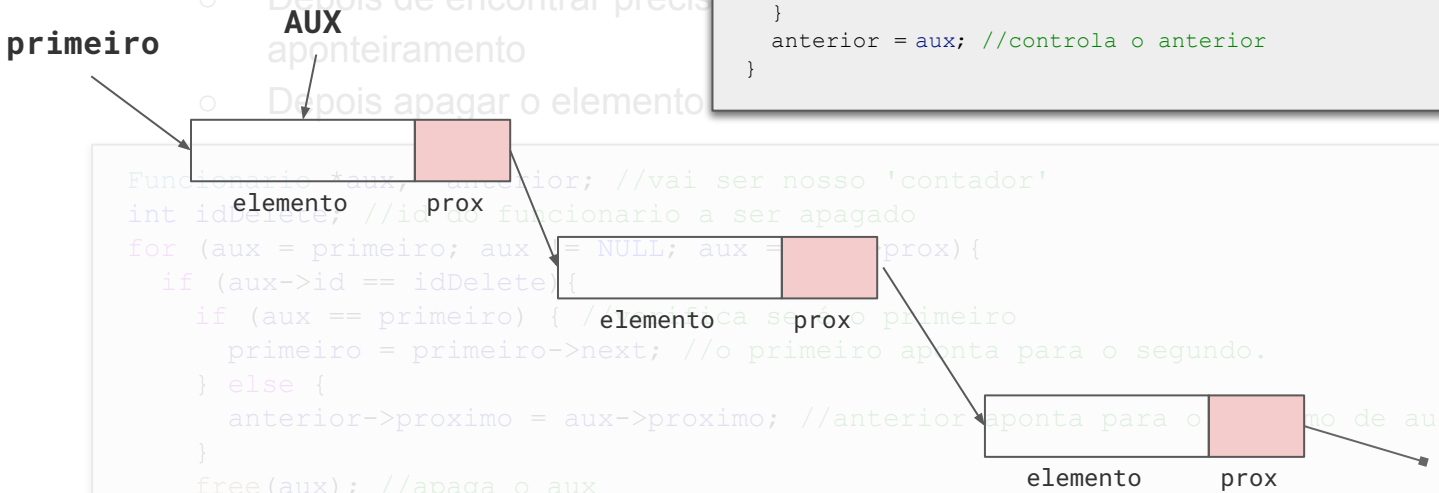
```c
struct funcionario{
  int id;
  char nome[TAM_NOME+1];
  double salario;
  struct funcionario *proximo;
}
typedef struct funcionario Funcionario;
```

```c
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

- Deletar um elemento da l
  - Precisa percorrer a lista at
  - Depois de encontrar precis
    aponteiramento
  - Depois apagar o elemento

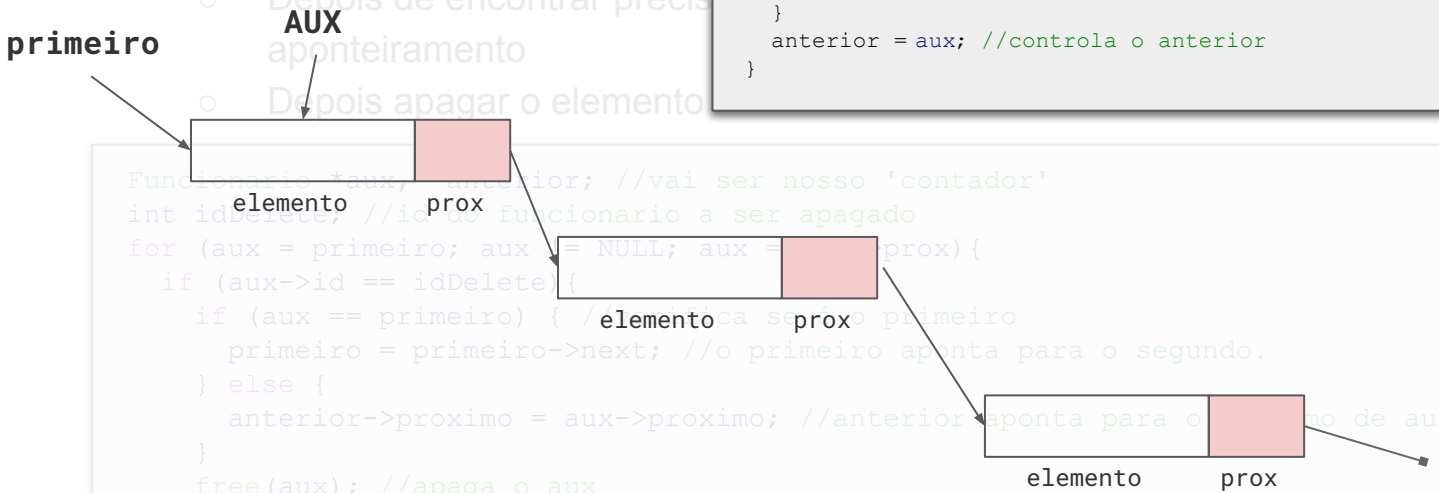
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta 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

- Deletar um elemento da l
  - Precisa percorrer a lista at
  - Depois de encontrar precis
    aponteiramento
  - Depois apagar o elemento

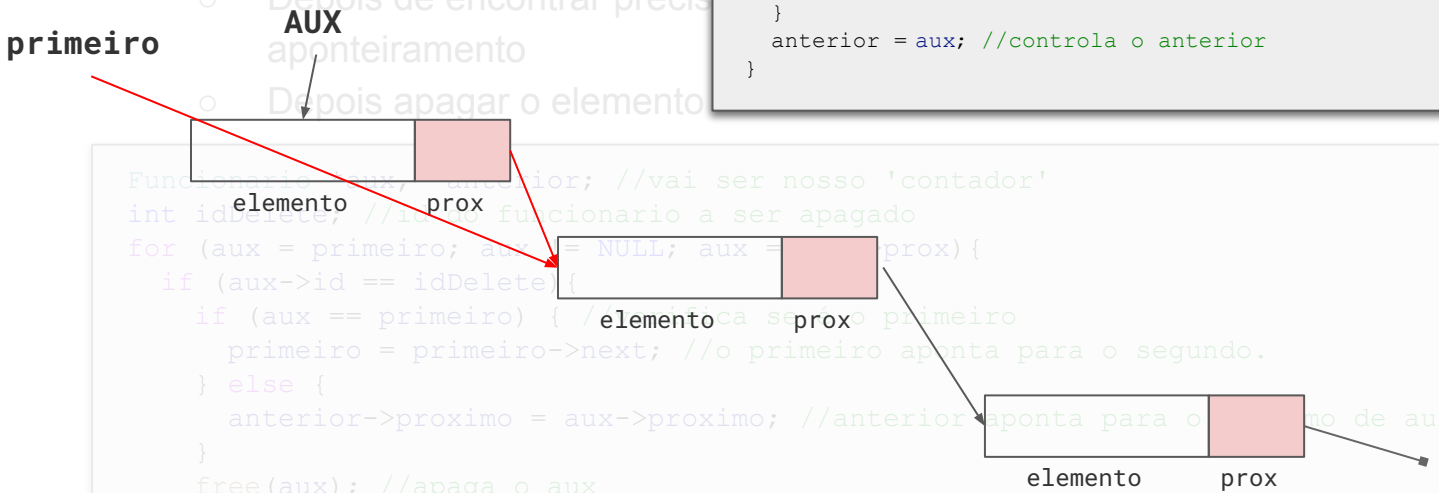
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta 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

- Deletar um elemento da l
  - Precisa percorrer a lista at
  - Depois de encontrar precis
    apont
  - Depois apagar o elemento

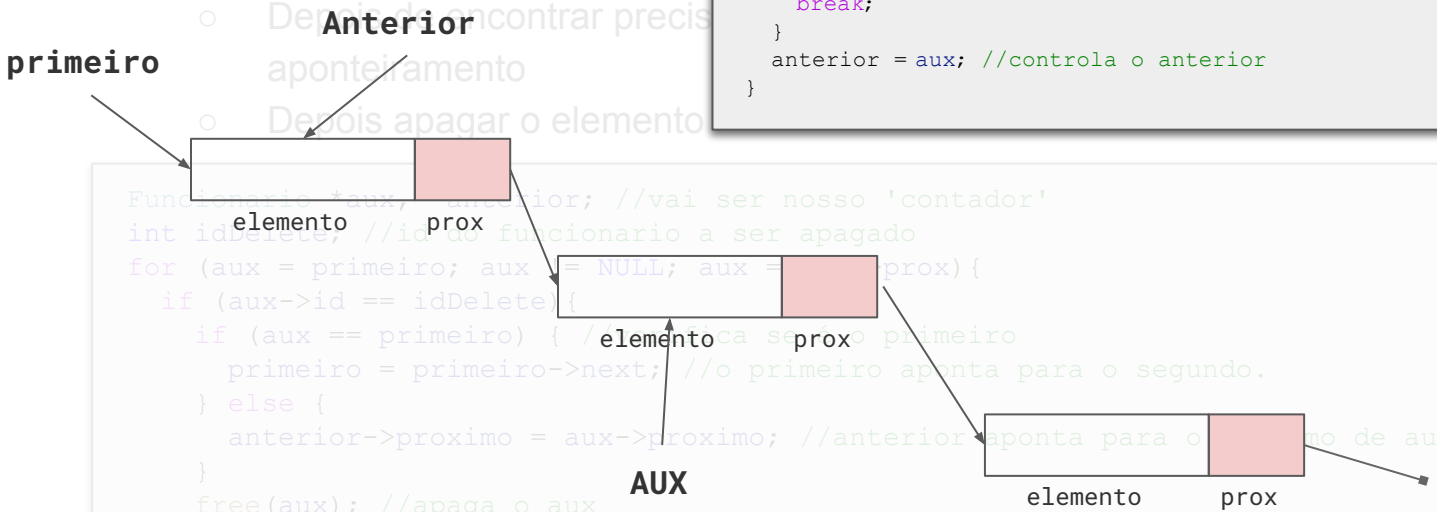
```c
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta 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

- Deletar um elemento da l
  - Precisa percorrer a lista at
  - Depois de encontrar precis
    apontamento
  - Depois apagar o elemento

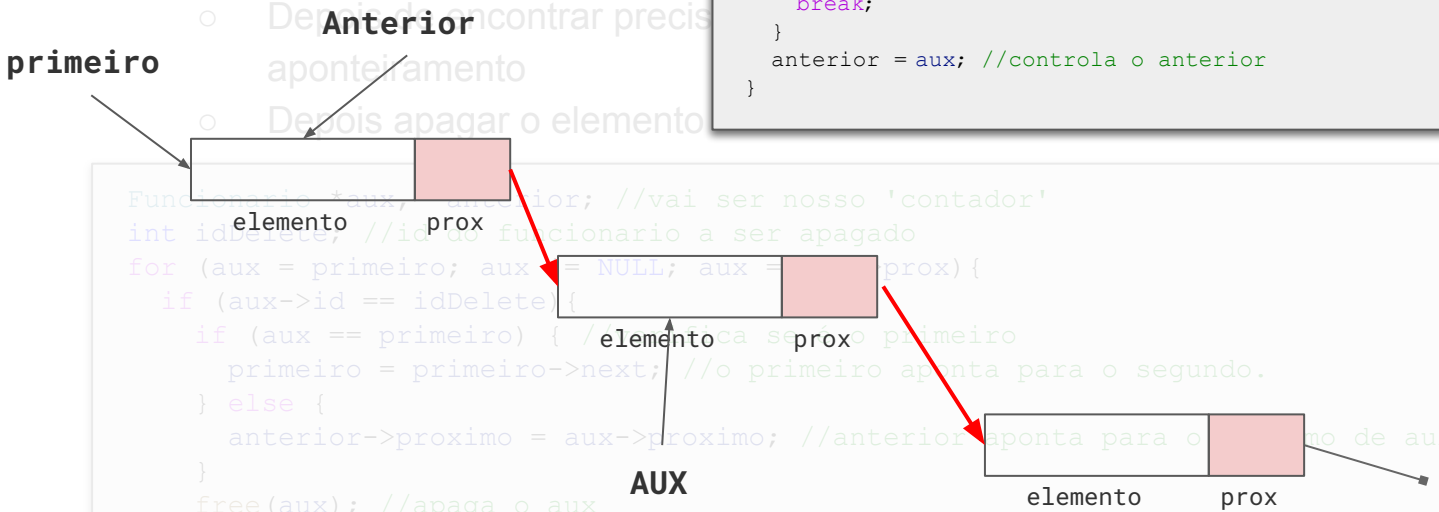
```c
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta 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

- Deletar um elemento da l
  - Precisa percorrer a lista at
  - Depois de encontrar precis
    apontamento
  - Depois apagar o elemento

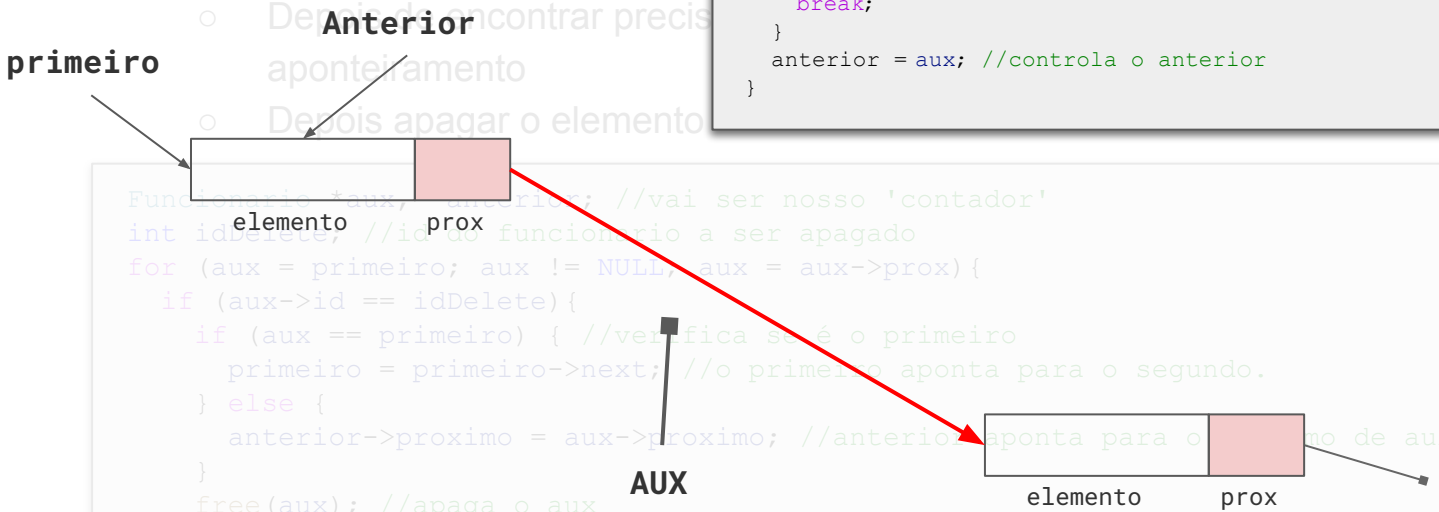
```c
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
  if (aux->id == idDelete){
    if (aux == primeiro) { //verifica se é o primeiro
      primeiro = primeiro->proximo; //o primeiro aponta 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

- Deletar um elemento da l
  - Precisa percorrer a lista at
  - Depois de encontrar precis
    apontamento
  - Depois apagar o elemento

```
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

- Como criamos a lista
  - Existe várias formas
  - Outra possibilidade

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

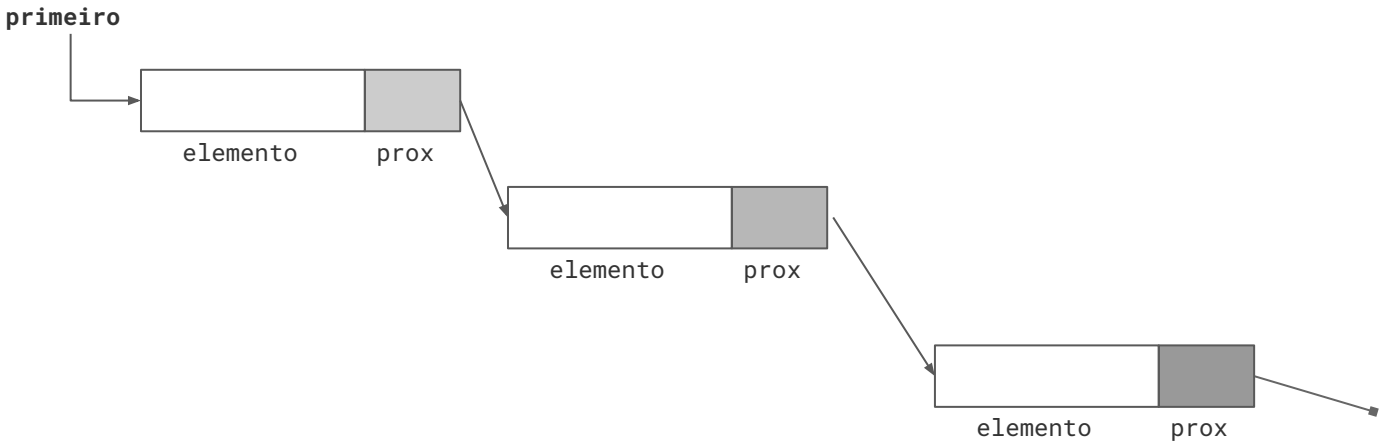
```c
typedef struct{
  Funcionario *primeiro;
} Lista;
```

Listas Duplamente encadeadas

Prof. Andrei Braga
Prof. Geomar Schreiner

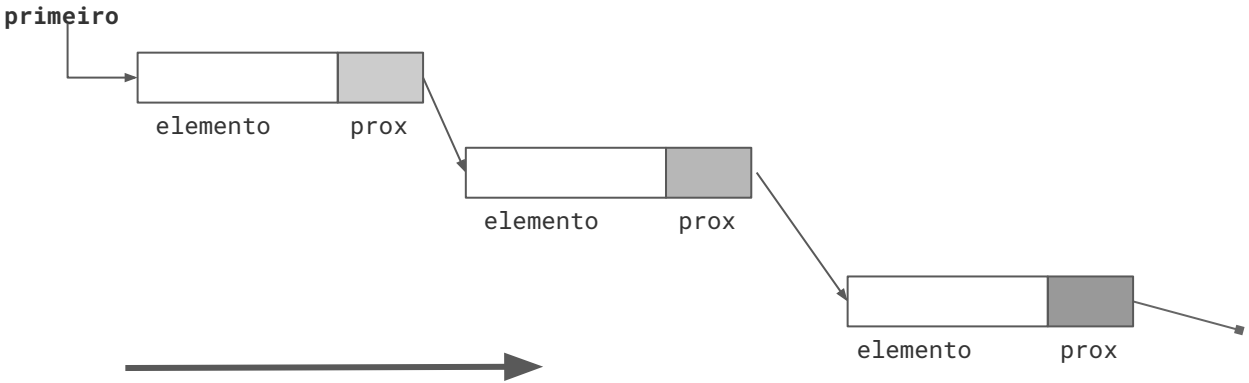
Listas

- Como faríamos para imprimir uma lista simplesmente encadeada em ordem inversa?



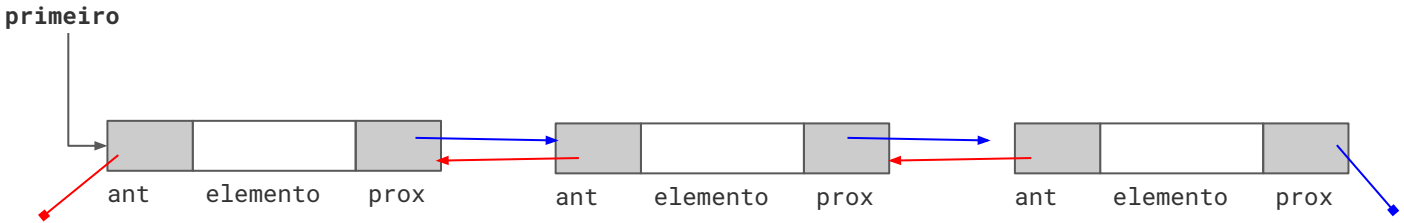
Listas

- Como faríamos para imprimir uma lista simplesmente encadeada em ordem inversa?
  - Uma lista simplesmente encadeada apenas nos permite o acesso a informação em uma direção
  - Não existe uma forma de fazer isso com uma performance legal
  - Deletar um elemento da lista também não é trivial já que precisamos armazenar o elemento anterior.



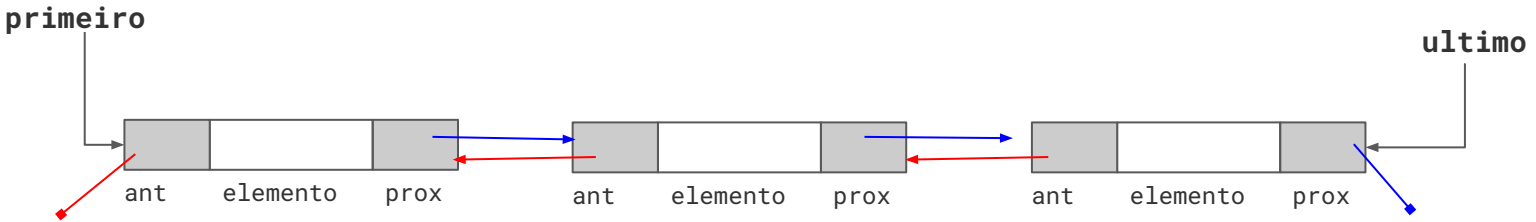
Listas

- Para resolver estes problemas podemos utilizar uma **estrutura** que **aponte** para o seu **próximo** mas também para o **anterior**.
- Esta estrutura é chamada de lista duplamente encadeada
  - Utilizando esta lista, sabemos facilmente o próximo elemento e o elemento anterior



Listas

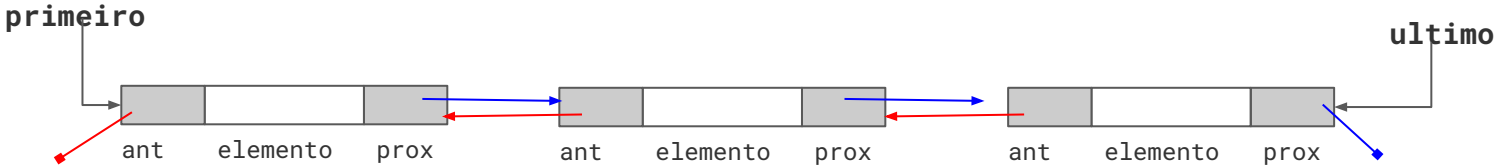
- Em uma lista duplamente encadeada cada elemento possui um ponteiro para seu anterior e um ponteiro para o seu próximo.
  - Utilizando esta lista, sabemos facilmente o próximo elemento e o elemento anterior
  - Já que temos a estrutura encadeada pelo próximo e pelo anterior podemos **armazenar** o primeiro (**head**) e o último (**tail**) elemento da lista



Listas

- Em uma lista duplamente encadeada cada elemento possui um ponteiro para seu anterior e um ponteiro para o seu próximo.
  - Já que temos a estrutura encadeada pelo próximo e pelo anterior podemos **armazenar** o primeiro (**head**) e o último (**tail**) elemento da lista

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

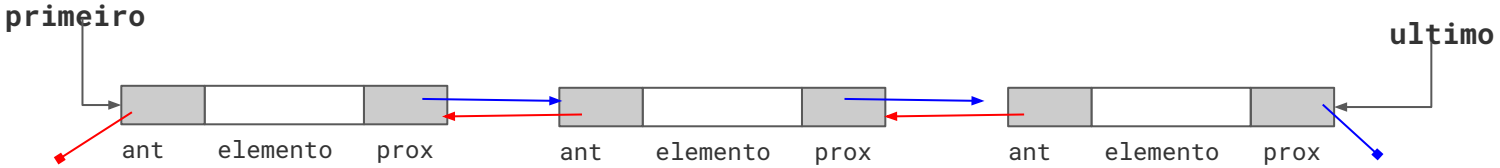


Listas

- Em uma lista duplamente encadeada cada elemento possui um ponteiro para seu anterior e um ponteiro para o seu próximo.
  - Já que temos a estrutura encadeada pelo próximo e pelo anterior podemos **armazenar** o primeiro (**head**) e o último (**tail**) elemento da lista

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

Sempre devemos ter cuidado para o elemento ou apontar para outro elemento ou NULL

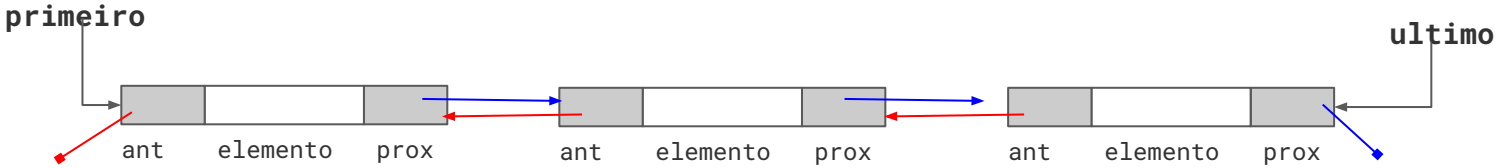


Listas

- Em uma lista duplamente encadeada cada elemento possui um ponteiro para seu anterior e um ponteiro para o seu próximo.
  - Já que temos a estrutura encadeada pelo próximo e pelo anterior podemos **armazenar** o primeiro (**head**) e o último (**tail**) elemento da lista

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

Sempre devemos ter cuidado para o elemento ou apontar para outro elemento ou NULL

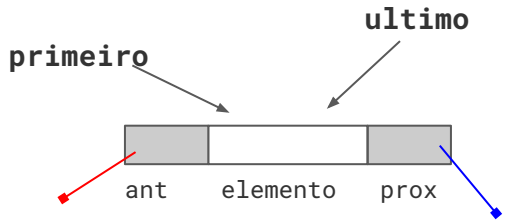


Listas

- Inserir um elemento

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *primeiro, *ultimo;
primeiro = malloc (sizeof(Funcionario));

primeiro->id = 1;
primeiro->idade = 31;
primeiro->salario = 234.0;
primeiro->proximo = NULL;
primeiro->anterior = NULL;

ultimo = primeiro;
```

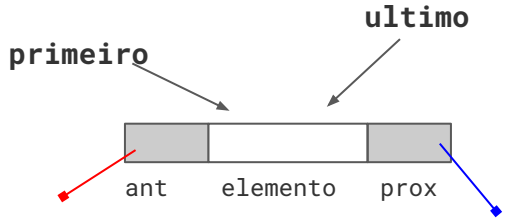


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - Fim da lista
  - No meio

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *primeiro, *ultimo;
primeiro = malloc (sizeof(Funcionario));

primeiro->id = 1;
primeiro->idade = 31;
primeiro->salario = 234.0;
primeiro->proximo = NULL;
primeiro->anterior = NULL;

ultimo = primeiro;
```

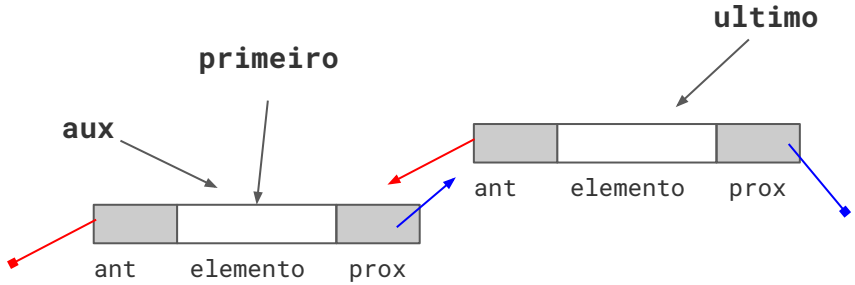


Listas

- Se eu adicionar mais um, onde adiciono
  - **Inicio da lista**
  - Fim da lista
  - No meio

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = primeiro;
  primeiro->anterior = aux;
  primeiro = aux;
}
```

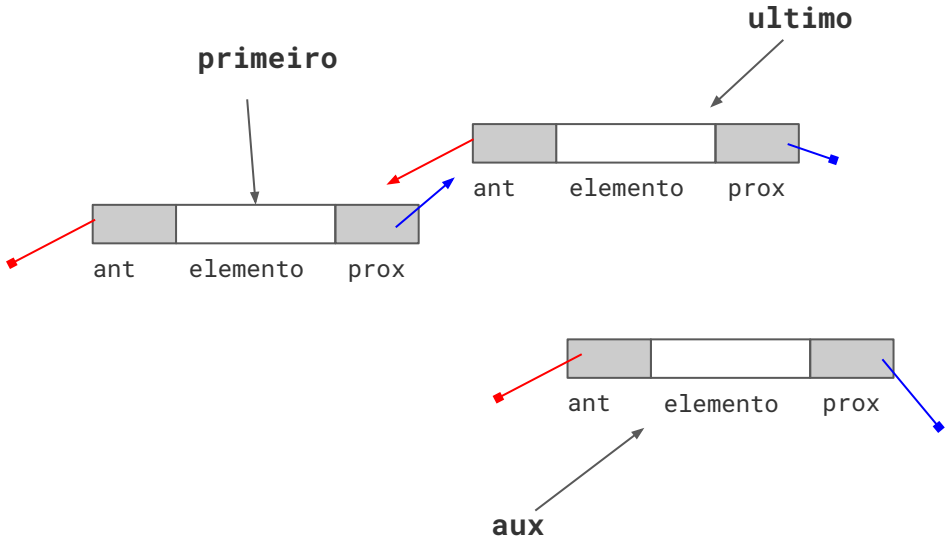


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - **Fim da lista**
  - No meio

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->anterior = ultimo;
  ultimo->proximo = aux;
  ultimo = aux;
}
```

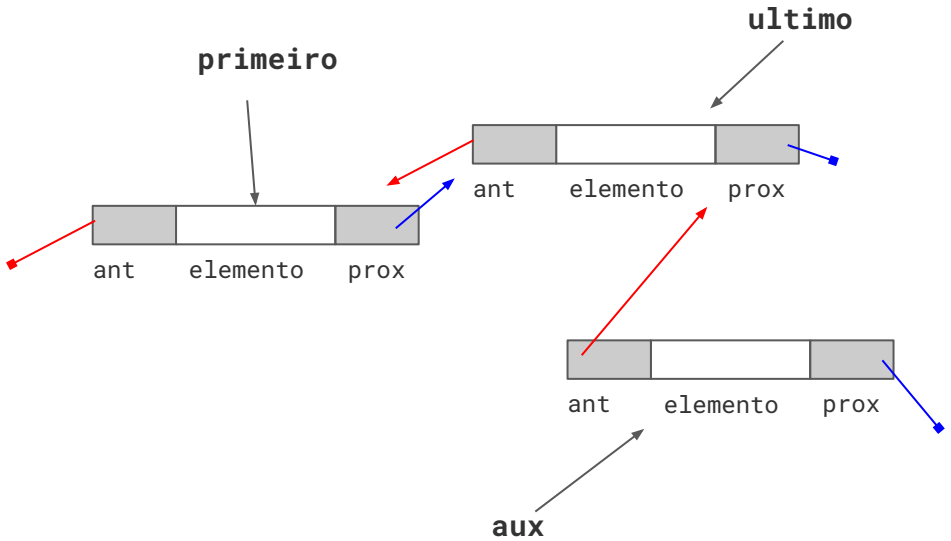


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - **Fim da lista**
  - No meio

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->anterior = ultimo;
  ultimo->proximo = aux;
  ultimo = aux;
}
```

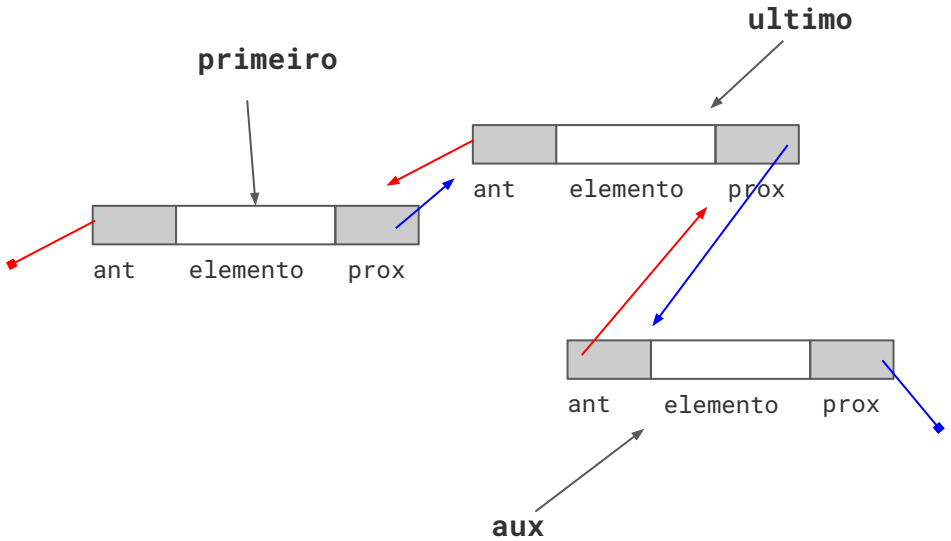


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - **Fim da lista**
  - No meio

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->anterior = ultimo;
  ultimo->proximo = aux;
  ultimo = aux;
}
```

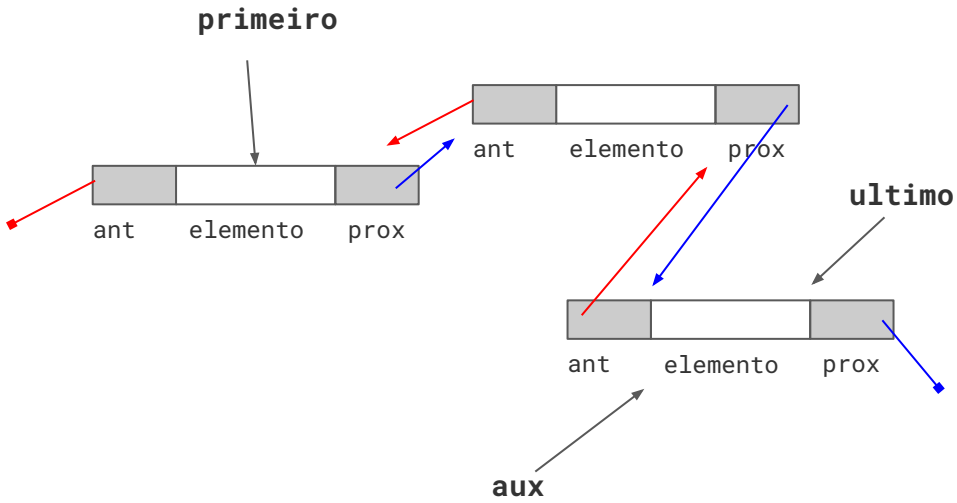


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - **Fim da lista**
  - No meio

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->anterior = ultimo;
  ultimo->proximo = aux;
  ultimo = aux;
}
```

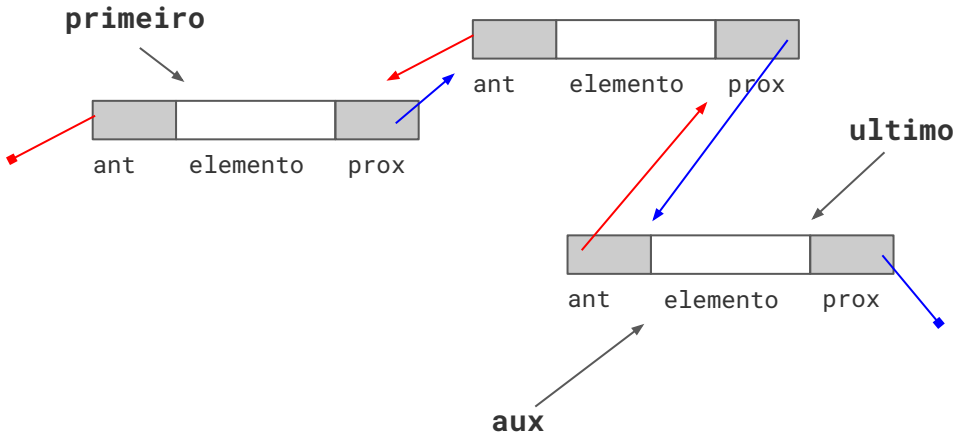


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - Fim da lista
  - **No meio**

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```

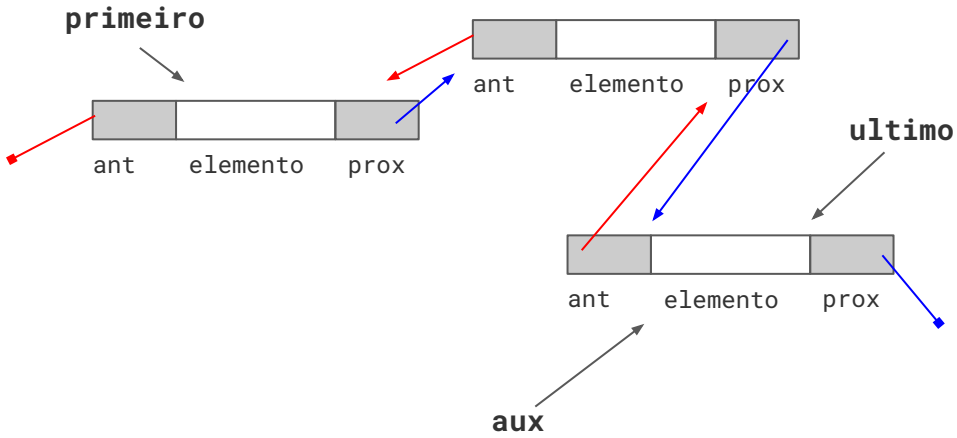


Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - Fim da lista
  - **No meio**

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

```c
Funcionario *aux;
for (i = 1; i< 10; i++){
  //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



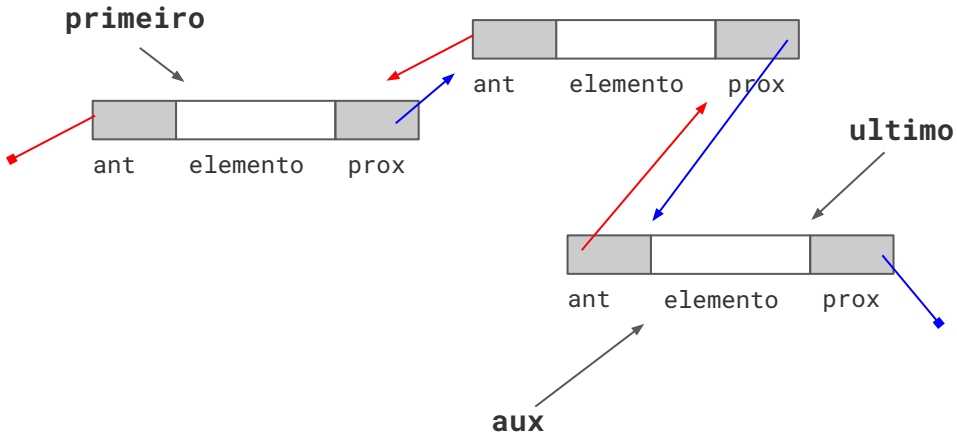
Listas

- Se eu adicionar mais um, onde adiciono
  - Inicio da lista
  - Fim
  - **No**

```c
struct funcionario{
  int id;
  int idade;
  double salario;
  struct funcionario *proximo;
  struct funcionario *anterior;
};
typedef struct funcionario Funcionario;
```

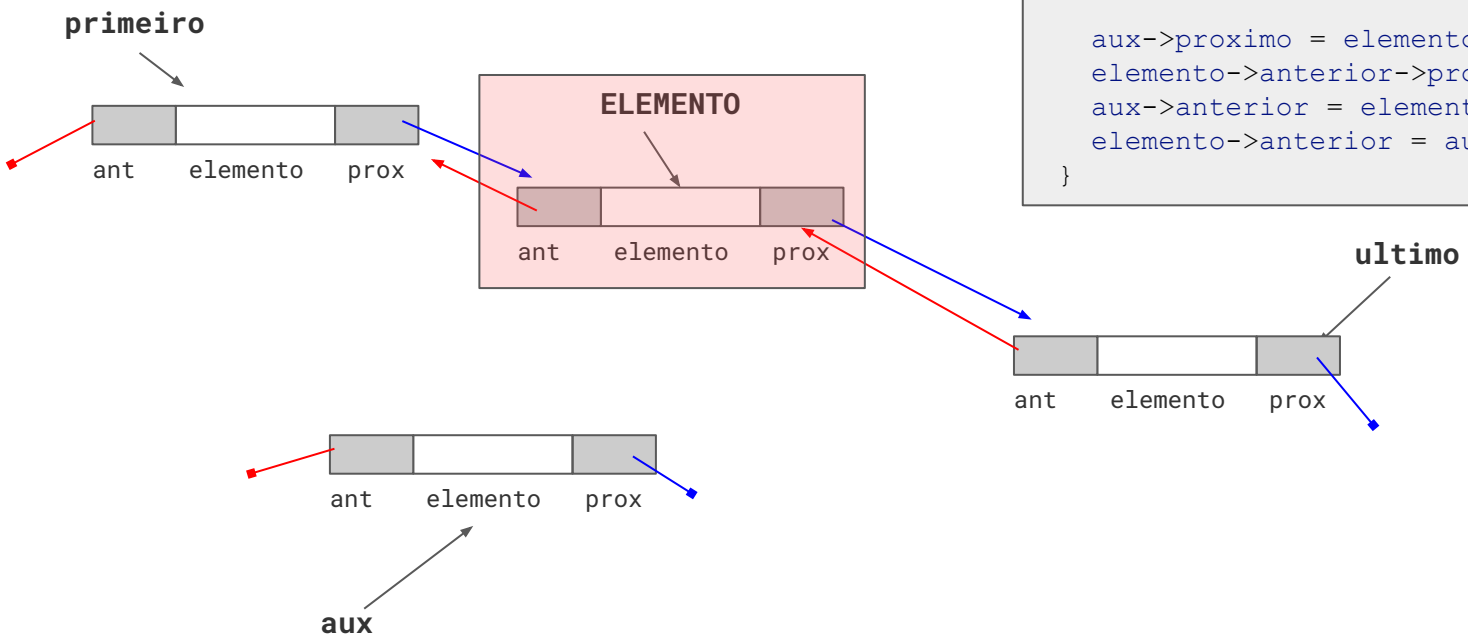
magia() não é uma função real, estamos usando para ilustrar que sabemos qual a posição correta.

```c
Funcionario *aux;
for (i = 1; i< 10;
  //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



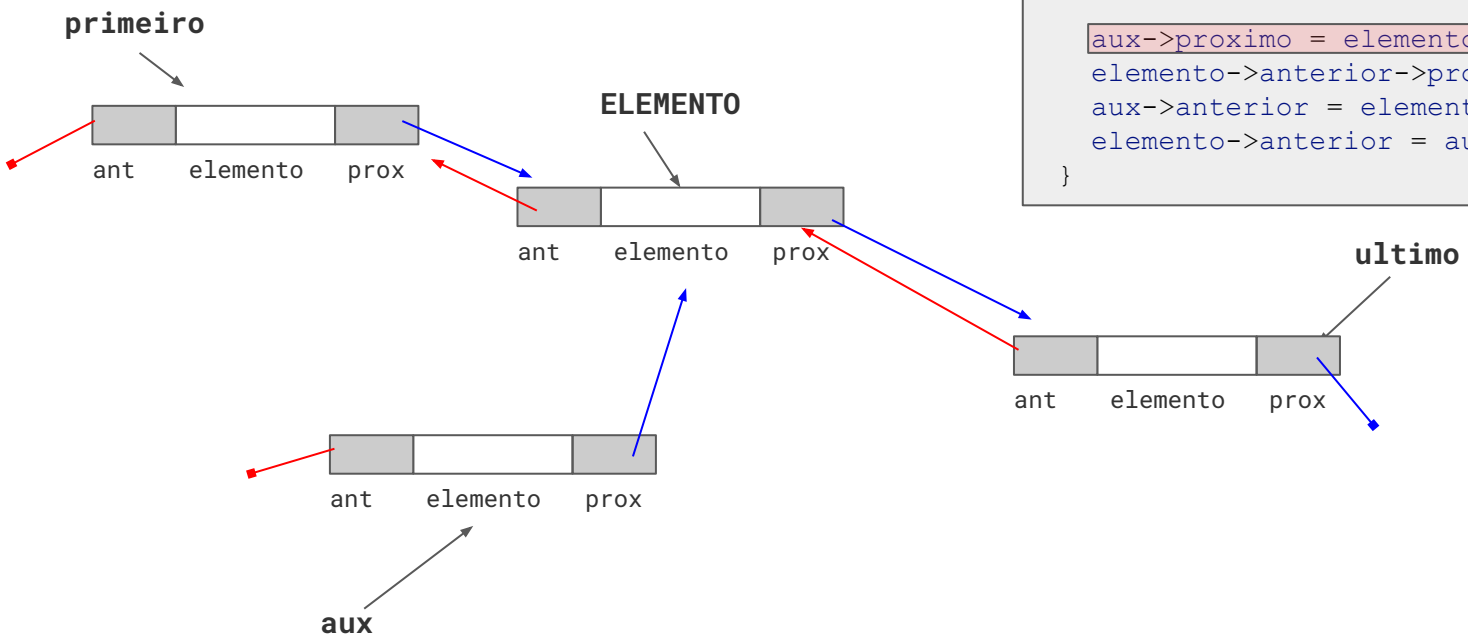
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
  //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



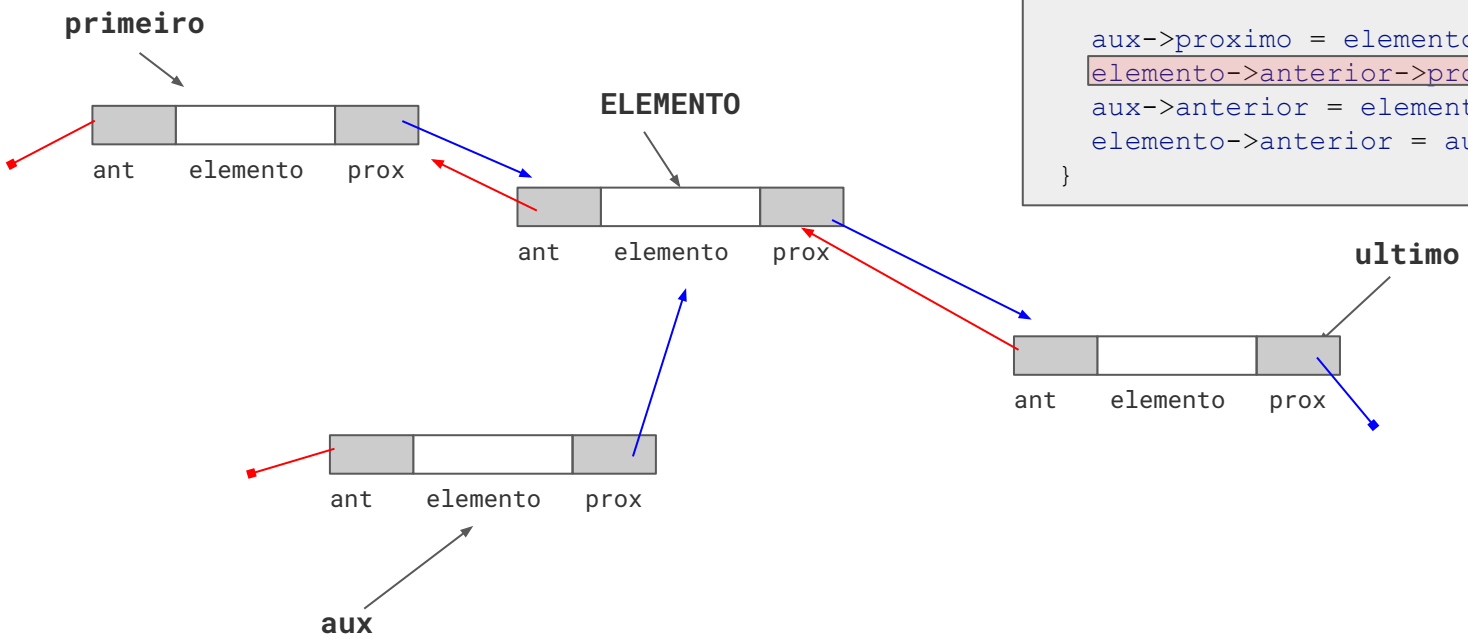
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



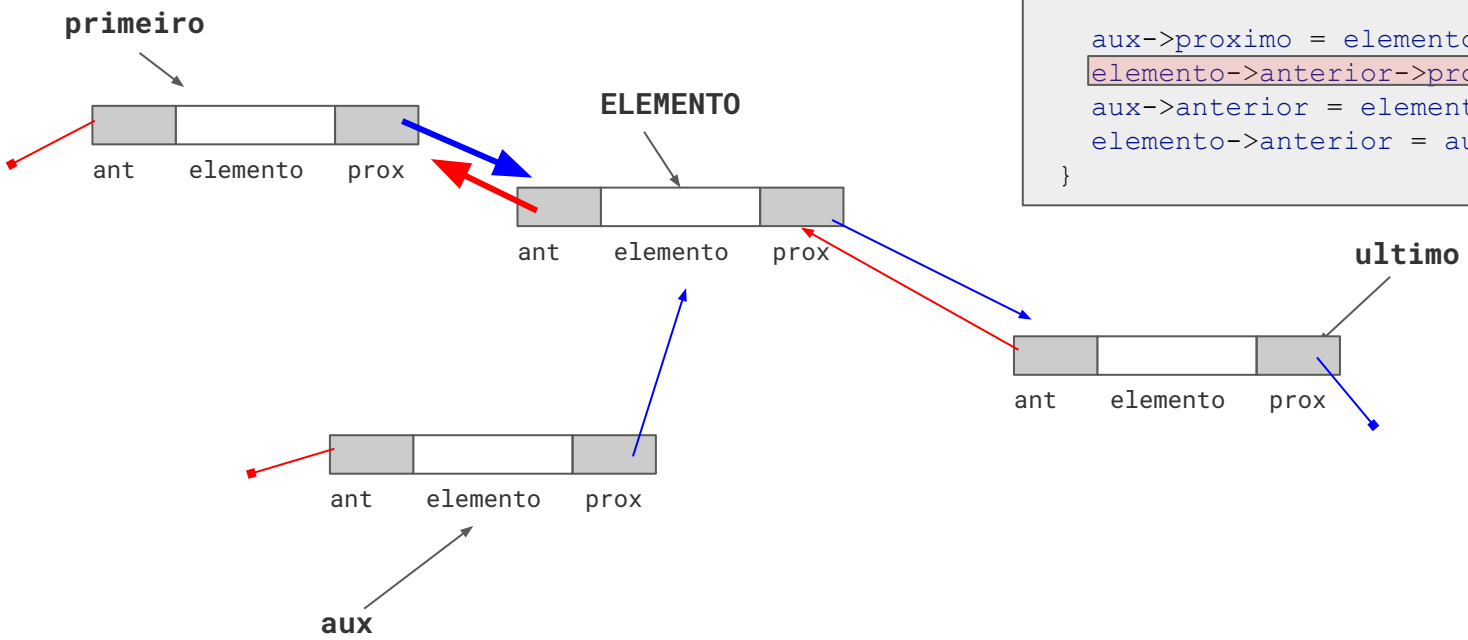
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



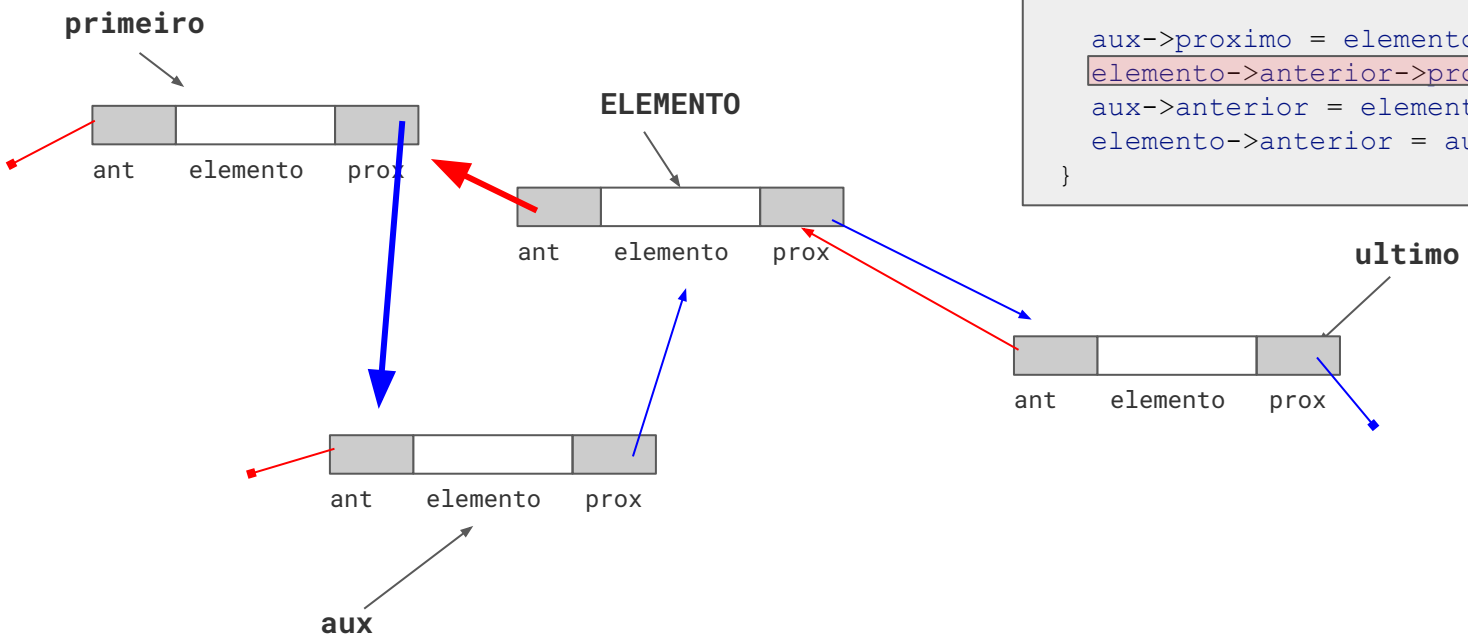
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



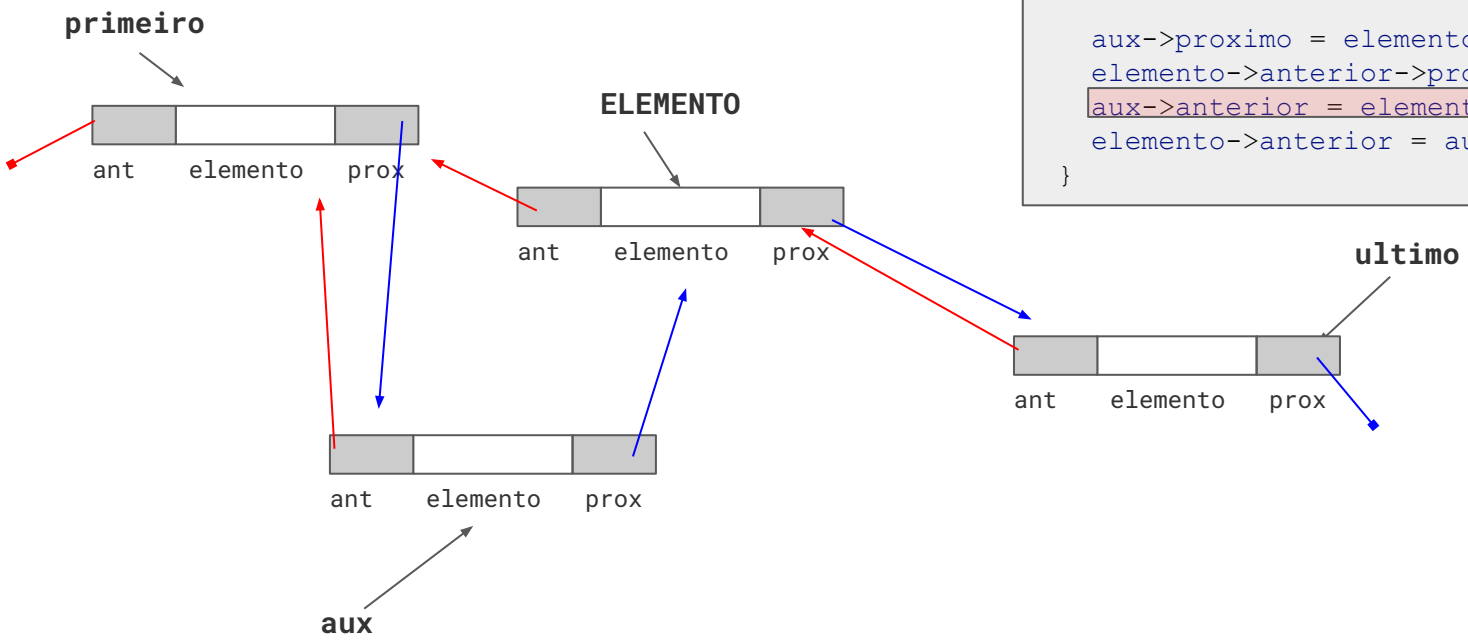
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



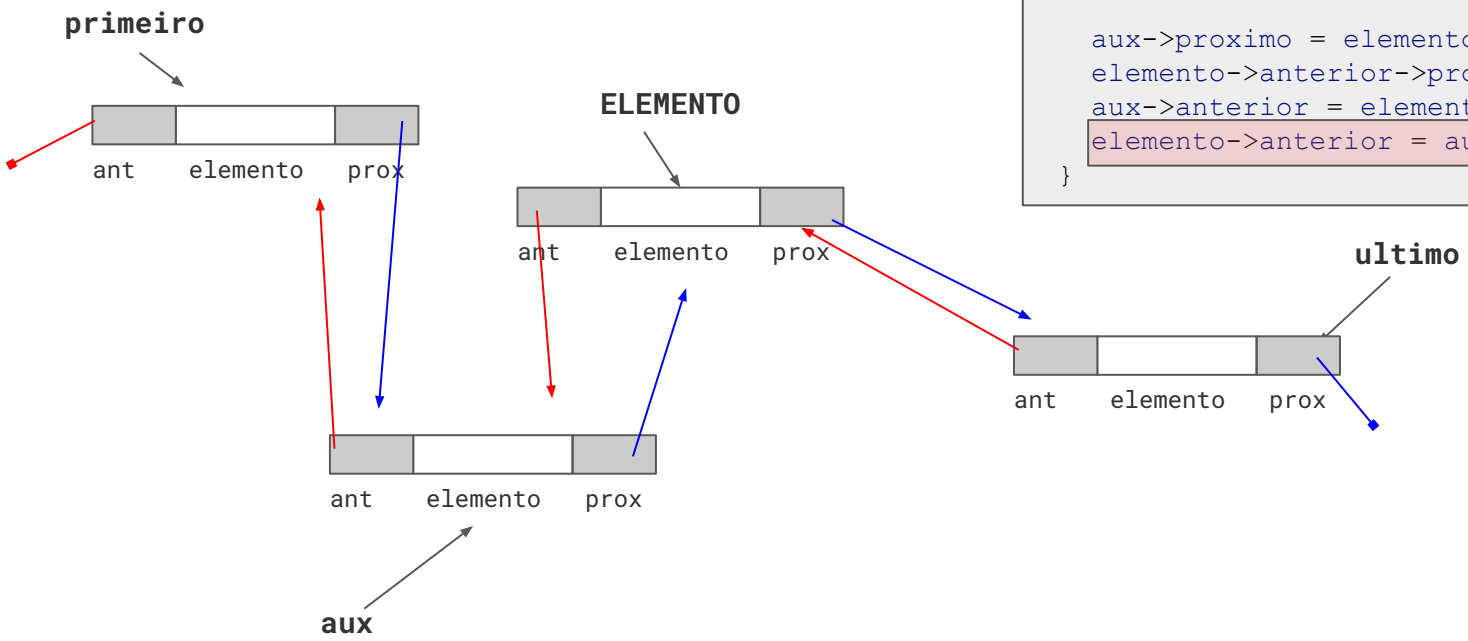
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



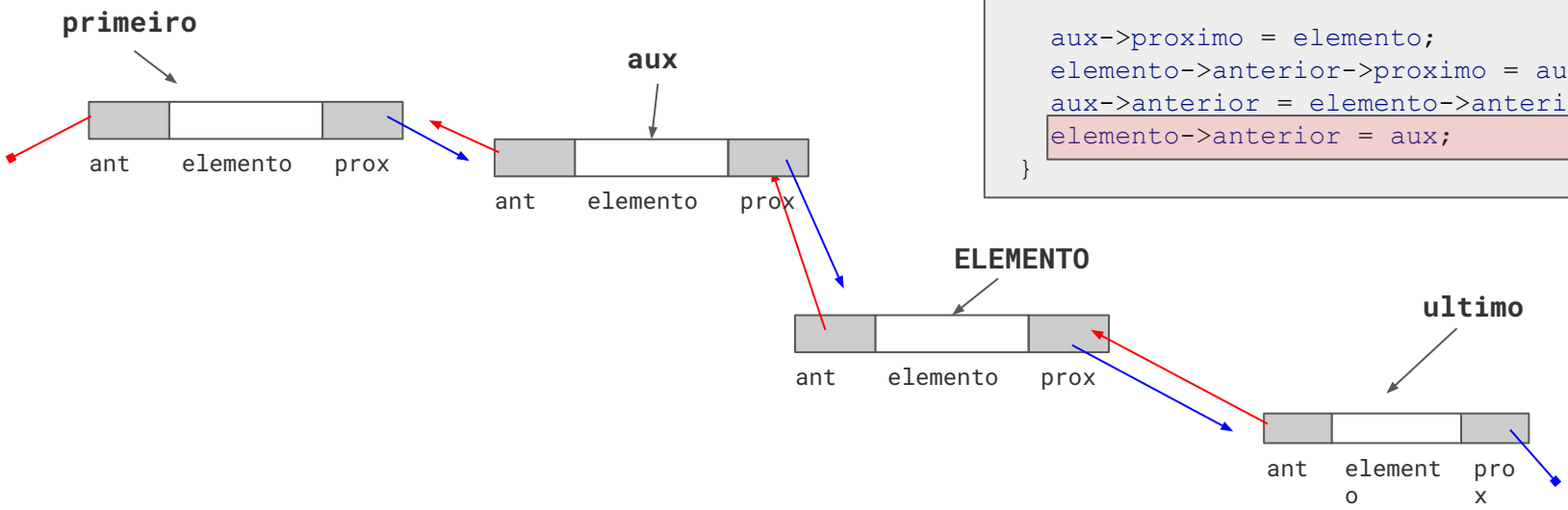
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



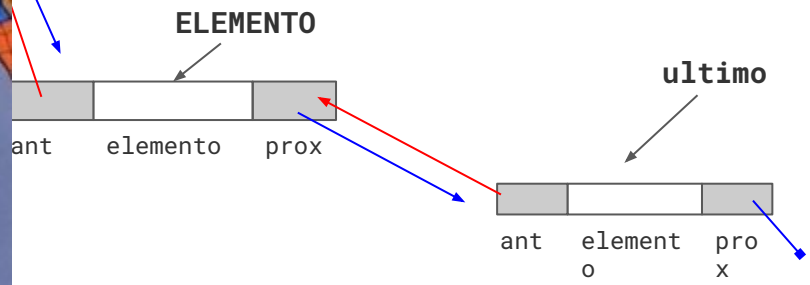
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



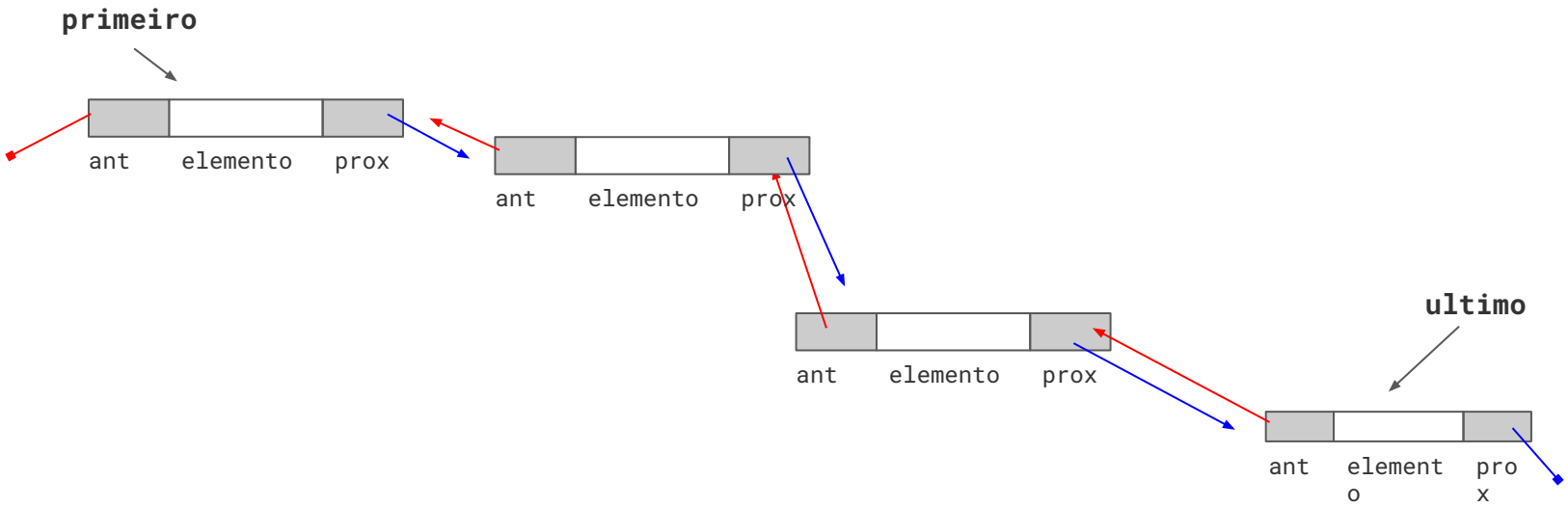
Listas

```
Funcionario *aux;
for (i = 1; i< 10; i++){
    //mágica que encontra a posição do elemento
  Funcionario *elemento = magia();
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 39;
  aux->salario = 234.0;
  aux->proximo = NULL;
  aux->anterior = NULL;

  aux->proximo = elemento;
  elemento->anterior->proximo = aux;
  aux->anterior = elemento->anterior;
  elemento->anterior = aux;
}
```



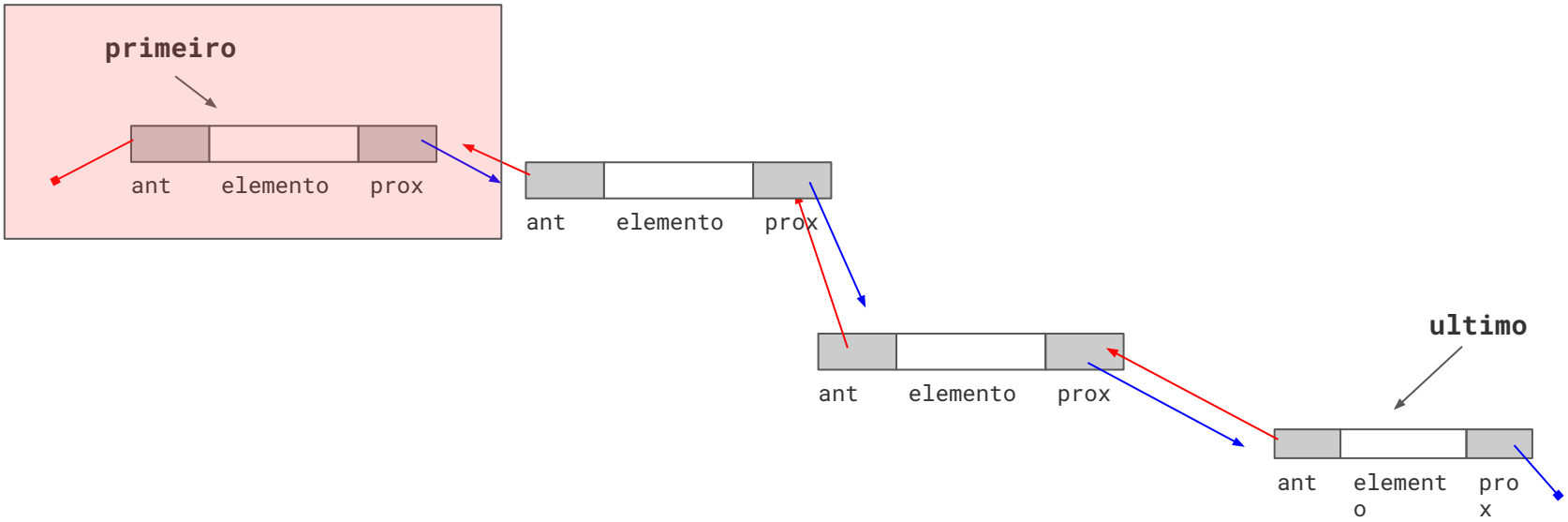
Listas

- Deletar um item



Listas

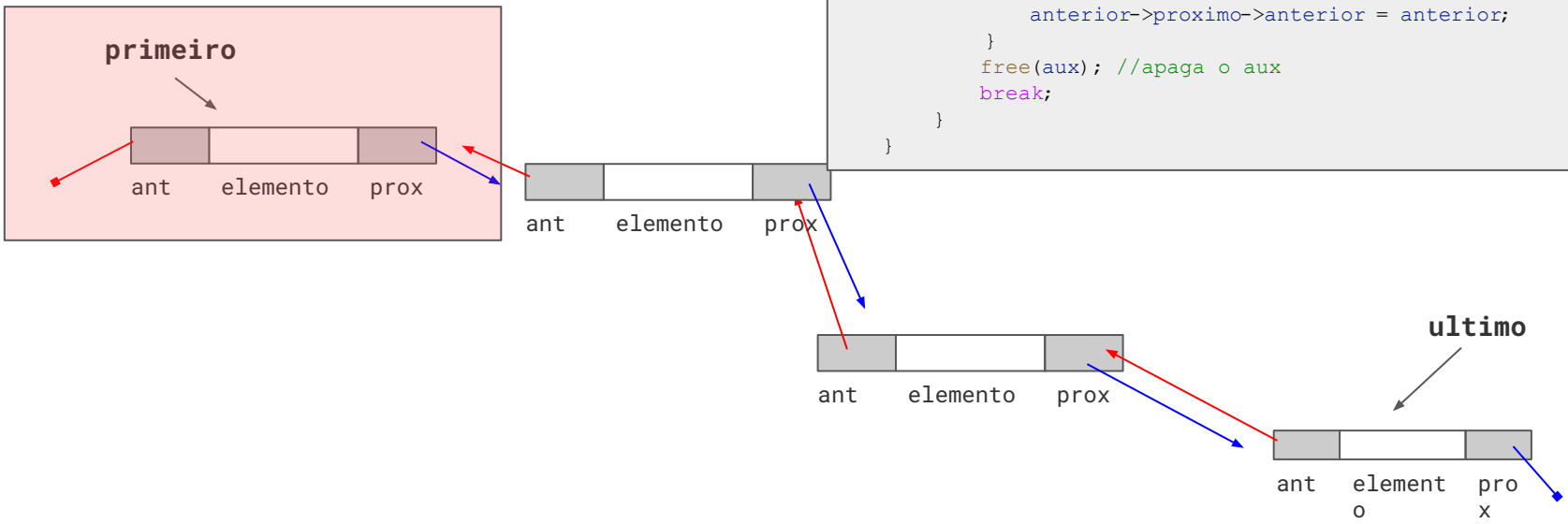
- Para eliminar um item devemos considerar 3 casos
  - Eliminar do início



Listas

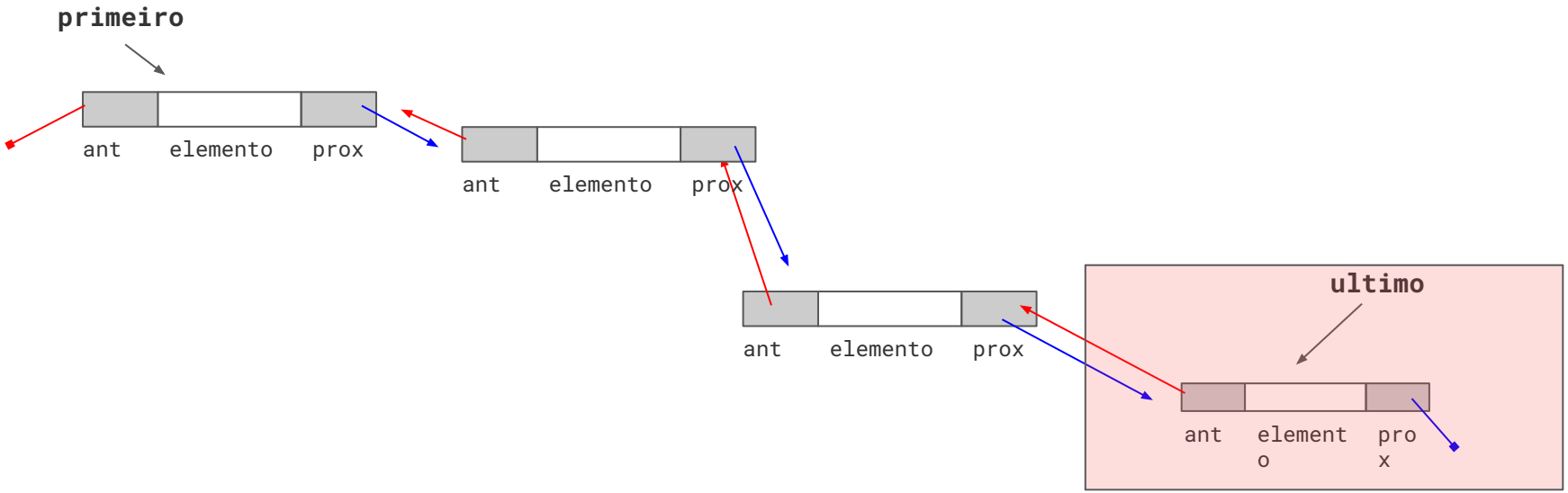
- Para eliminar um item devemos cons
  - Eliminar do início

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



Listas

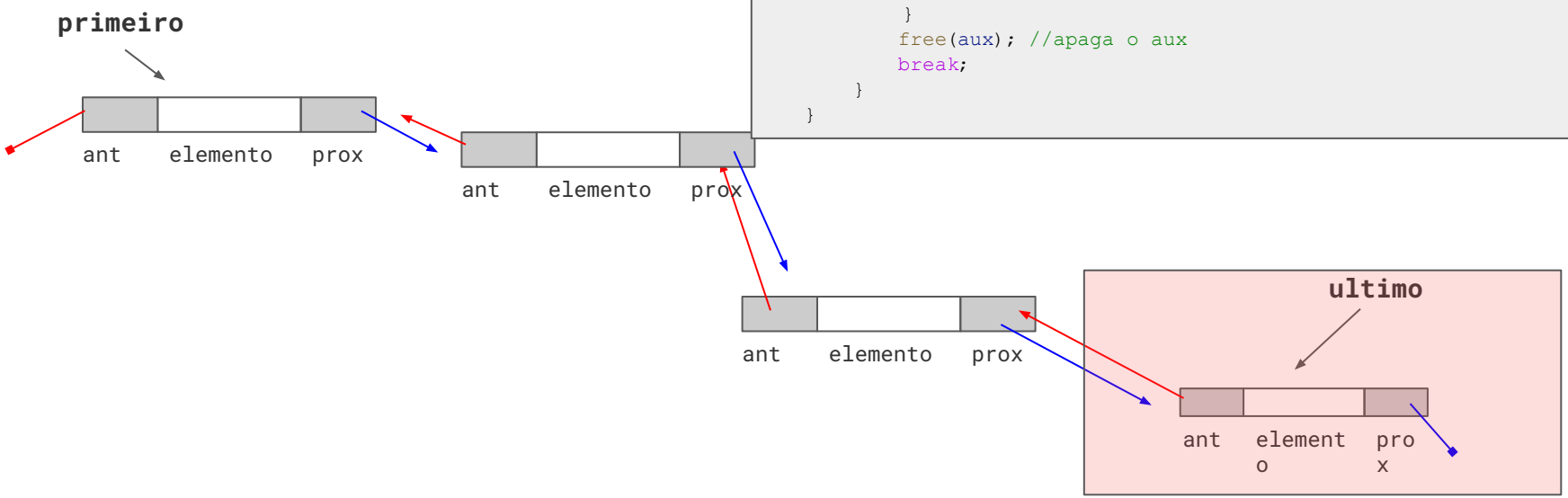
- Para eliminar um item devemos considerar 3 casos
  - Eliminar do início
  - Eliminar do fim



Listas

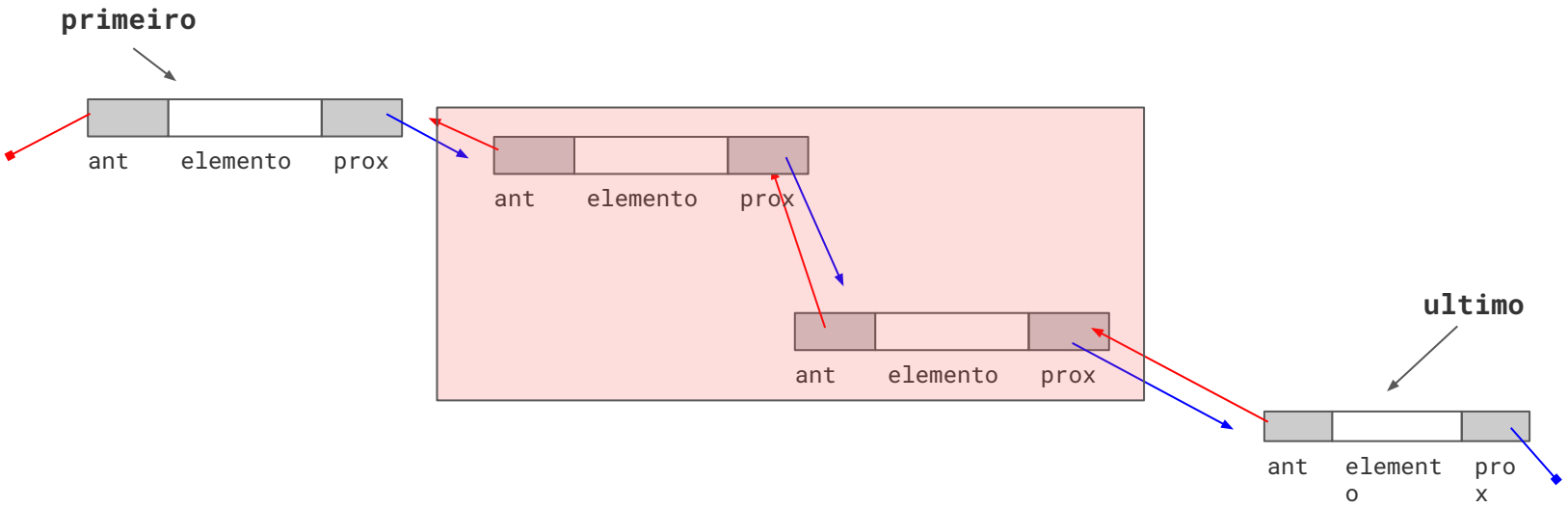
- Para eliminar um item devemos cons
  - Eliminar do início
  - Eliminar do fim

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



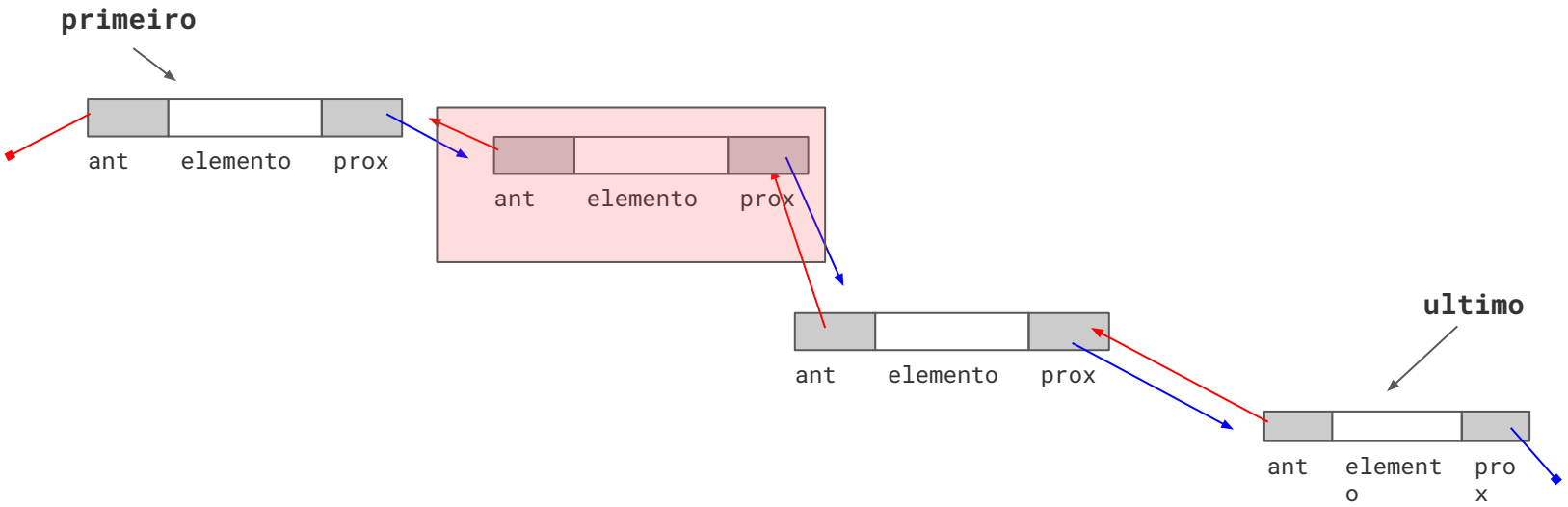
Listas

- Para eliminar um item devemos considerar 3 casos
  - Eliminar do início
  - Eliminar do fim
  - Eliminar do meio



Listas

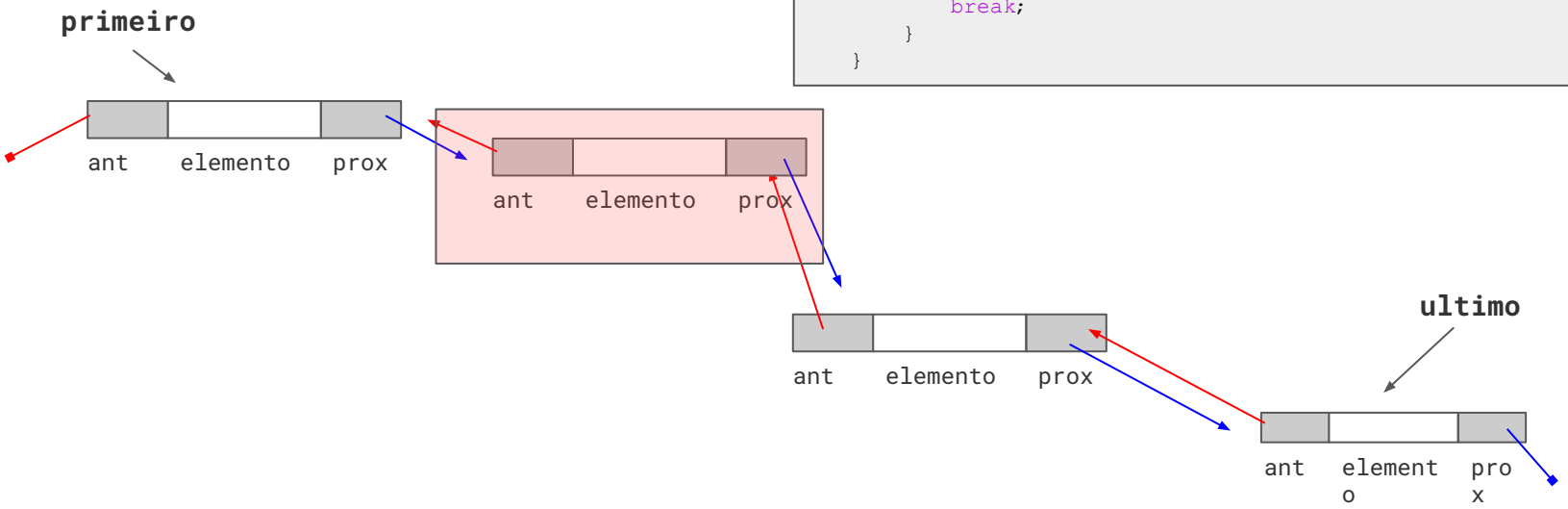
- Para eliminar um item devemos considerar 3 casos
  - Eliminar do início
  - Eliminar do fim
  - Eliminar do meio



Listas

- Para eliminar um item devemos cons
  - Eliminar do início
  - Eliminar do fim
  - Eliminar do meio

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



Exercícios

1. Considerando as definições a seguir, faça o que é pedido nos itens abaixo:

```c
typedef struct {
  int dia;
  int mes;
  int ano;
} Data;
```

```c
struct funcionario{
  int id;
  char nome[41];
  double salario;
  Data nascimento;
  struct funcionario *proximo;
};
typedef struct funcionario Funcionario;
```

   a. Crie as estruturas indicadas, e crie o primeiro funcionário da lista;
   b. Adicione um segundo funcionário ao início da lista;
   c. Crie uma função capaz de imprimir todos os funcionários;

Exercícios

2. Considerando a estrutura proposta no exercício anterior, faça as seguintes adaptações em seu programa:
    a. O programa deve ler (do teclado) um inteiro N que representará o número de registros que o usuário irá inserir. Após a leitura seu programa deve ler os dados dos N registros e os inserir na lista encadeada.
    b. Imprima a lista para ver se todos os elementos estão presentes
    c. Faça uma função que deleta um funcionário. A função deve receber como parâmetro a lista, e o id do funcionário a ser deletado, e deve retornar o primeiro elemento da lista

Exercícios

3. Implemente uma função que receba um vetor de valores inteiros com N elementos e construa uma lista encadeada armazenando os elementos do vetor (elemento a elemento). Assim, se for recebido por parâmetro o vetor v[4] = {1,21,4,6} a função deve retornar uma lista encadeada onde o primeiro elemento é ‘1’, o segundo o ‘21’, o terceiro o ‘4’ e assim por diante. A função deve ter a seguinte assinatura: *ListaInt* *constroiLista (int n, int *v);

Exercícios

3. Implemente uma função que receba um vetor de valores inteiros com N elementos e construa uma lista encadeada armazenando os elementos do vetor (elemento a elemento). Assim, se for recebido por parâmetro o vetor v[4] = {1,21,4,6} a função deve retornar uma lista encadeada onde o primeiro elemento é ‘1’, o segundo o ‘21’, o terceiro o ‘4’ e assim por diante. A função deve ter a seguinte assinatura: *ListaInt* *constroiLista (int n, int *v);

Exercícios

4. Transforme a estrutura da lista implementada nas questões 1 e 2 em uma lista duplamente encadeada. E implemente as seguintes funcionalidades:
    a. Imprimir a lista do primeiro para o último elemento, e depois do último para o primeiro.
    b. Crie uma função de busca que apresenta as informações de um funcionário. A busca deve ser feita utilizando o id.
    c. Atualize a função de delete.