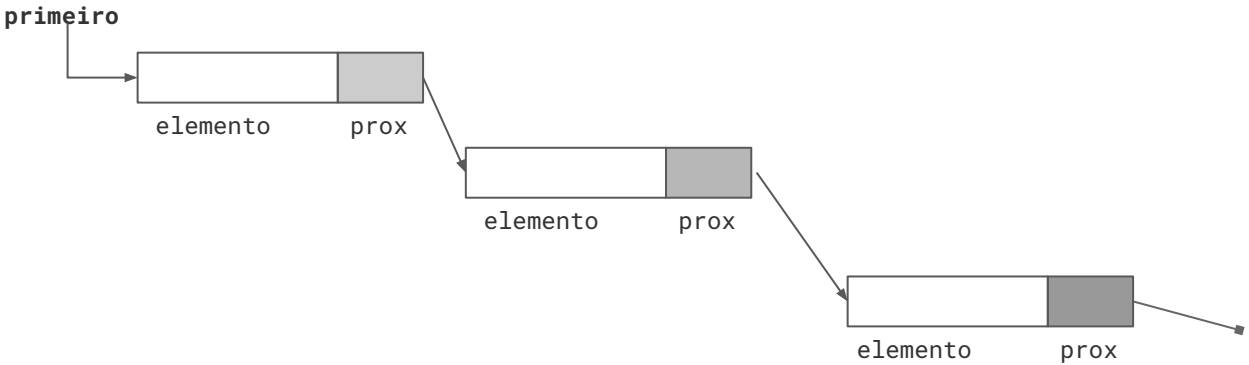
Listas Duplamente Encadeadas

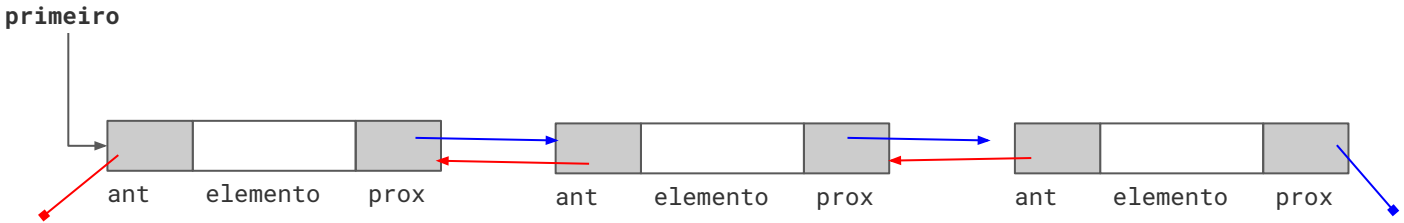
Lista Simplesmente Encadeada

- 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 boa performance
  - Deletar um elemento da lista também não é trivial, já que precisamos armazenar um ponteiro para o elemento anterior.



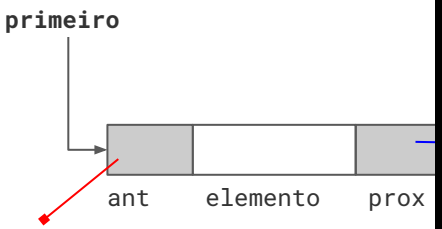
Lista Duplamente Encadeada

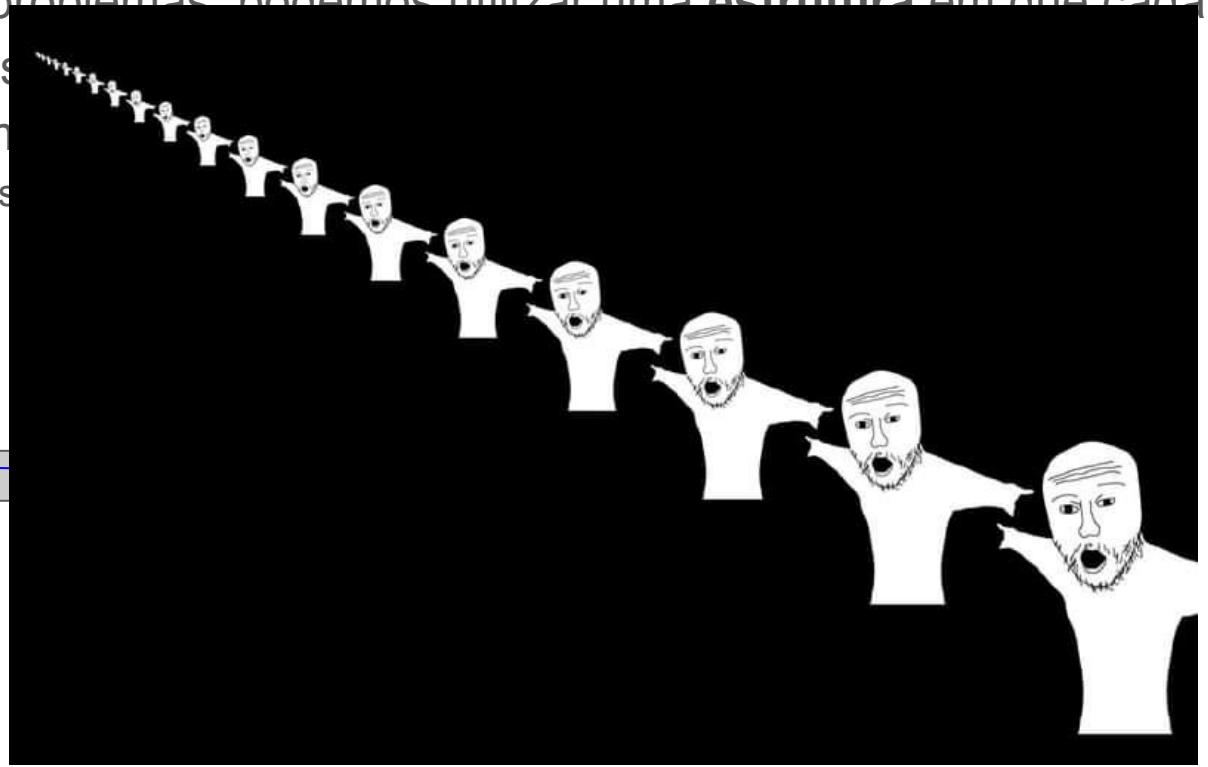
- Para resolver estes problemas, podemos utilizar uma **estrutura** em que cada nodo **aponte** para o seu **próximo** e também para o **anterior**.
- Esta estrutura é chamada de **lista duplamente encadeada**
  - A lista pode ser percorrida nos dois sentidos



Lista Duplamente Encadeada

- Para resolver estes problemas, podemos utilizar uma **estrutura** em que cada nodo **aponte** para o s
- Esta estrutura é chan
  - Utilizando esta lista, s

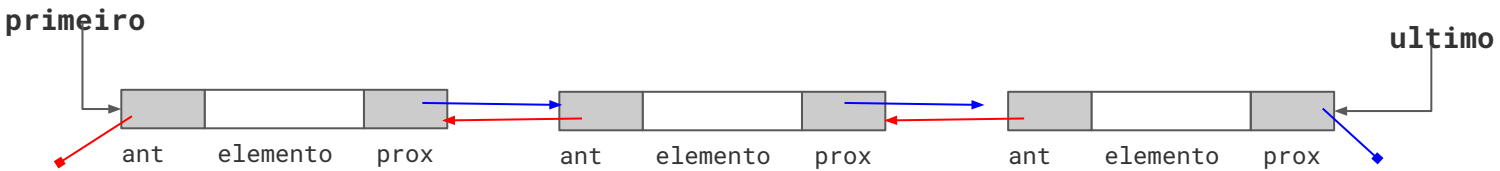




Lista Duplamente Encadeada

- 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;
```

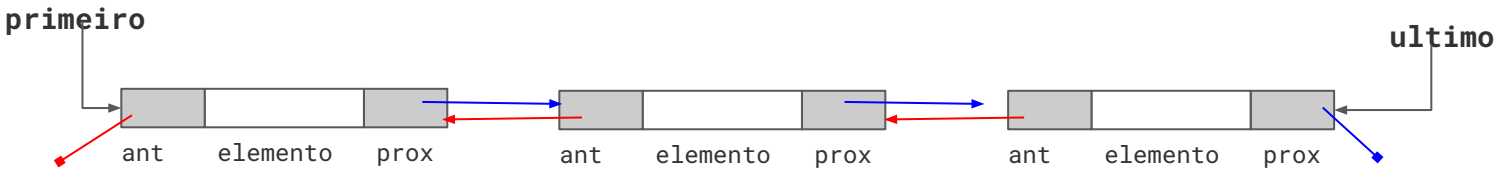


Lista Duplamente Encadeada

- 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



Lista Duplamente Encadeada

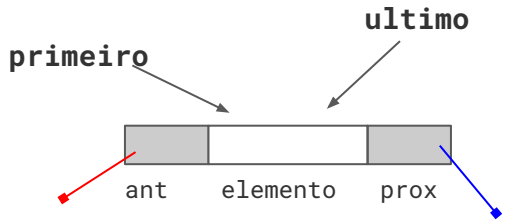
- Inserir o primeiro elemento

```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;
```



Lista Duplamente Encadeada

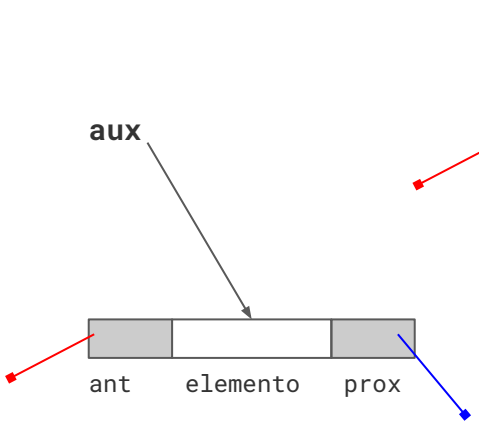
- Inserir novos elementos
  - Início da lista
  - Fim da lista
  - No meio

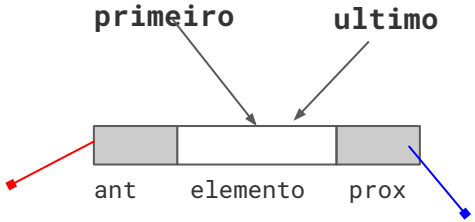
Lista Duplamente Encadeada

- Inserir novos elementos
  - **Início da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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



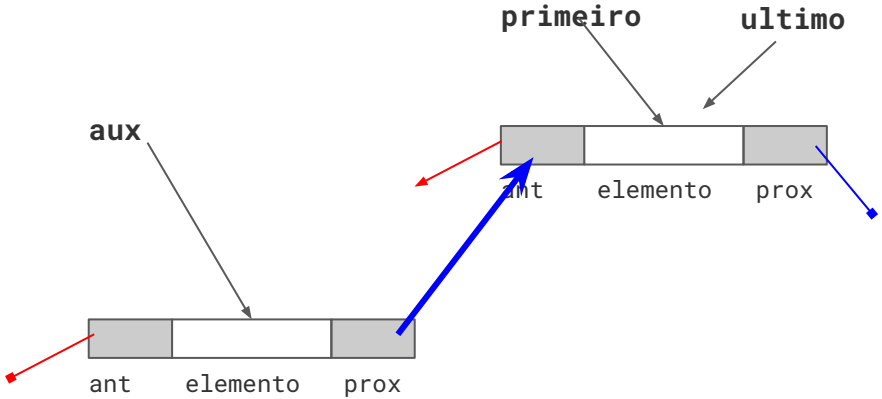


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Início da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

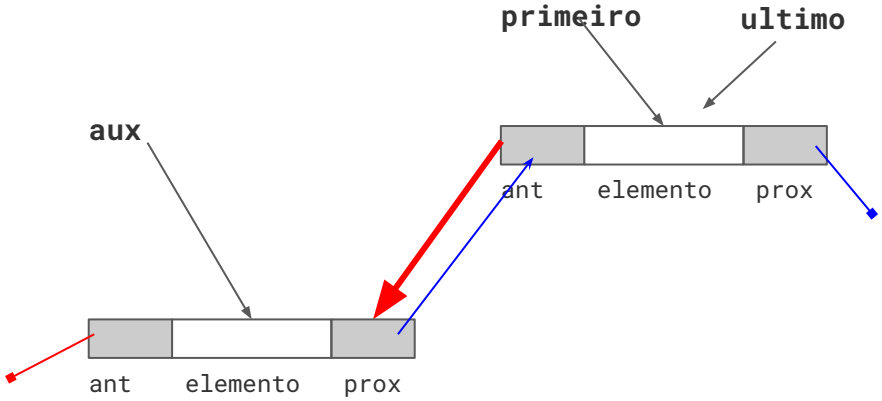


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Início da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

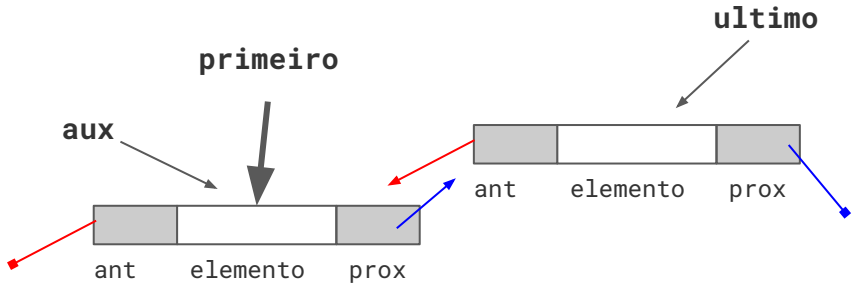


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Início da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

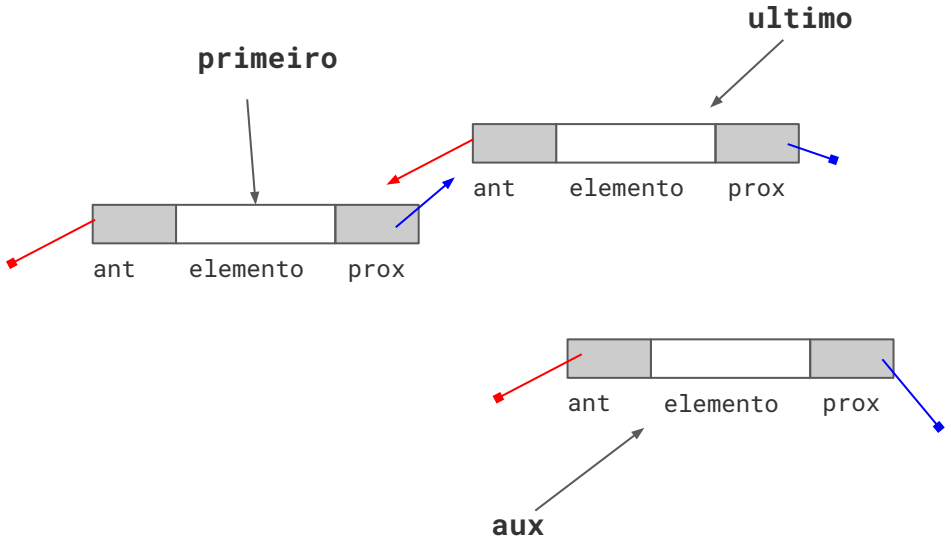


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Fim da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

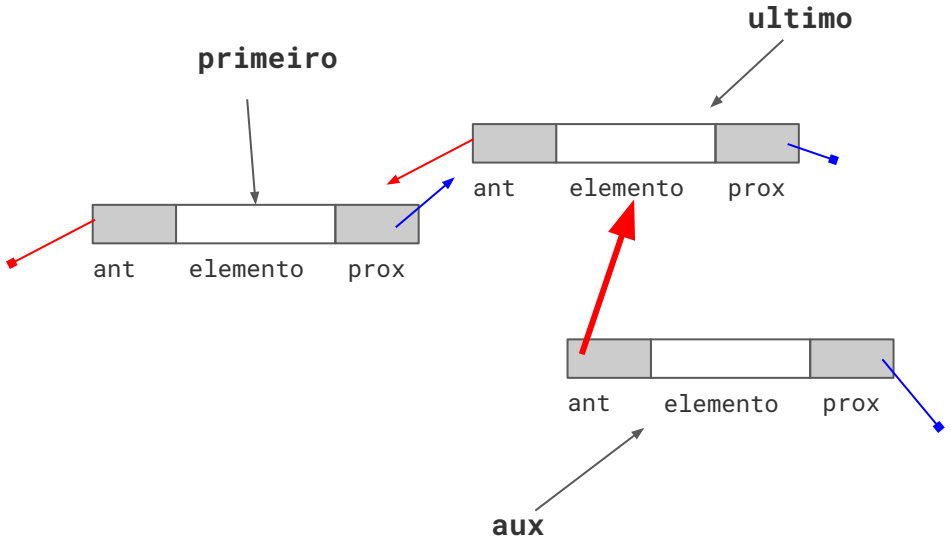


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Fim da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

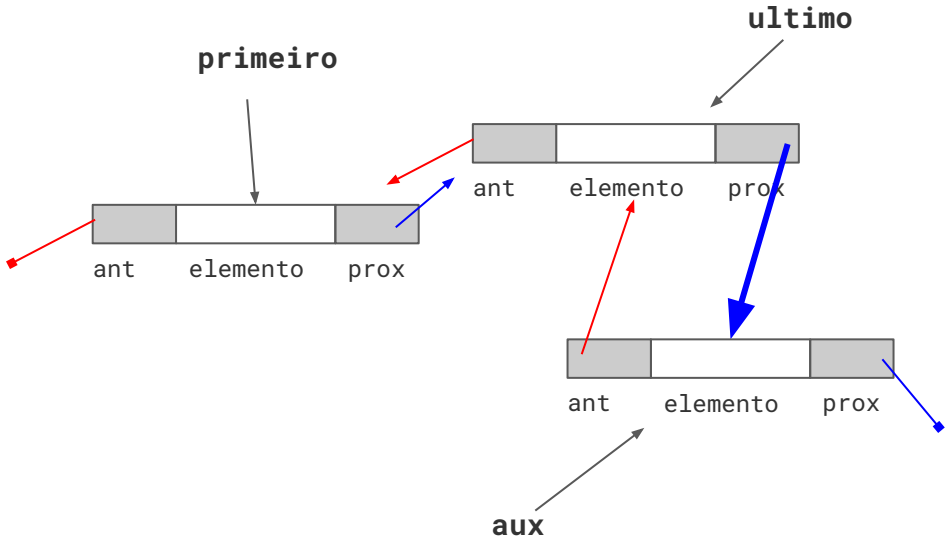


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Fim da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

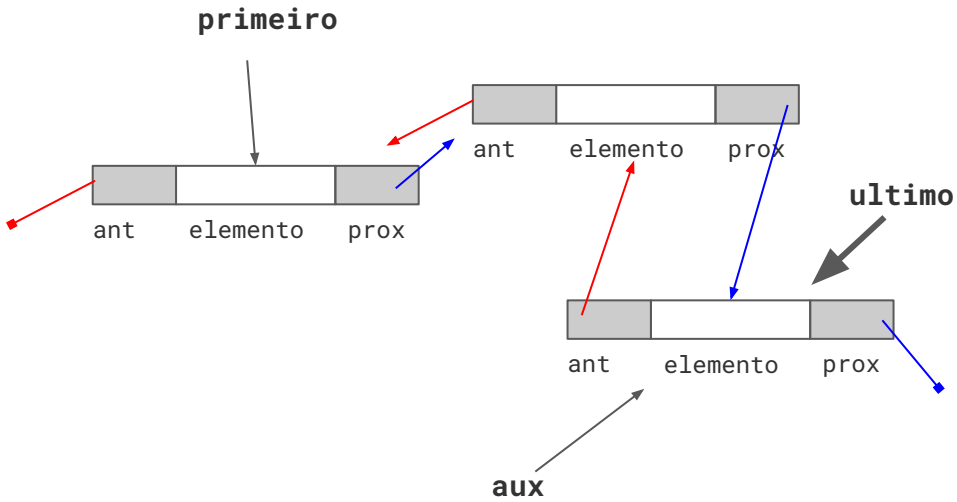


Lista Duplamente Encadeada

- Inserir novos elementos
  - **Fim da lista**

```
Funcionario *aux;
for (i = 1; i < 10; i++){
  aux = malloc (sizeof(Funcionario));
  aux->id = i+1;
  aux->idade = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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



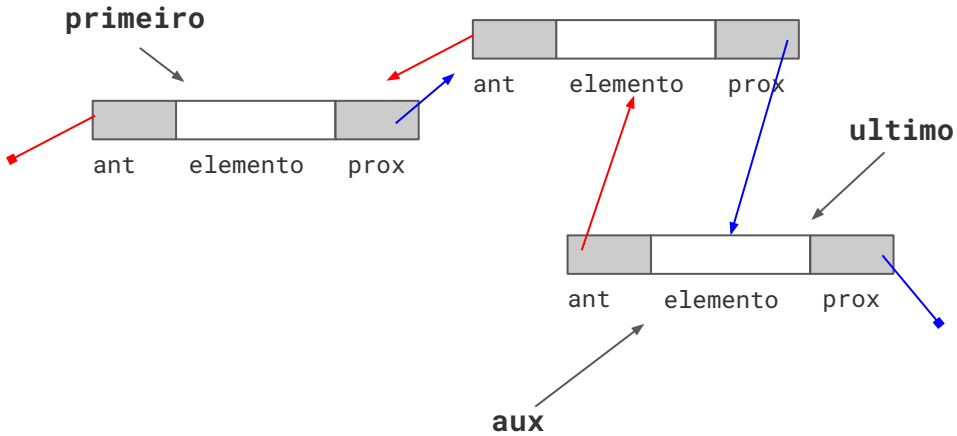
Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

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

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

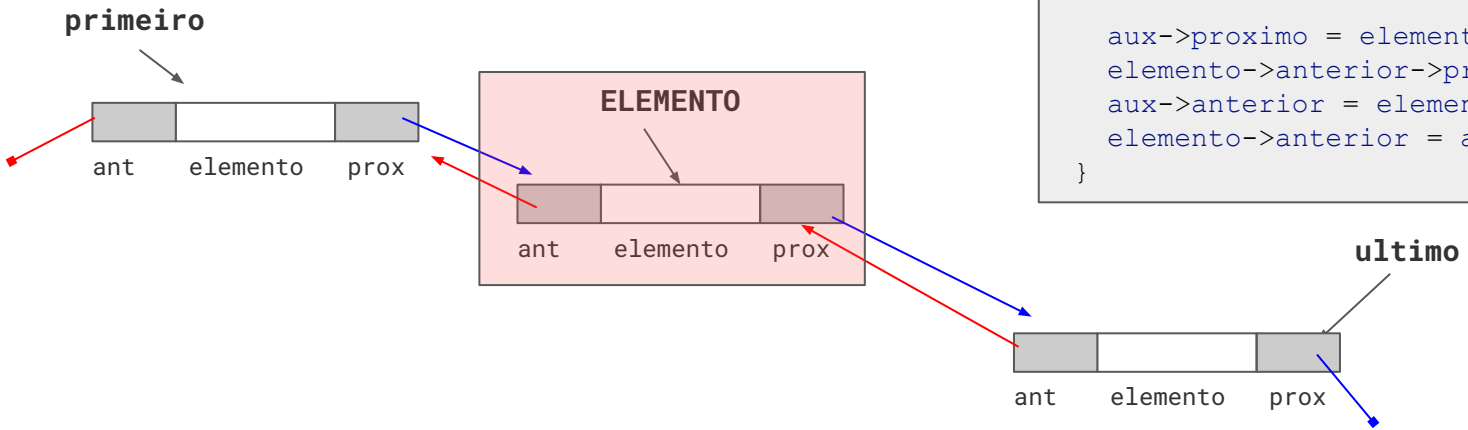


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

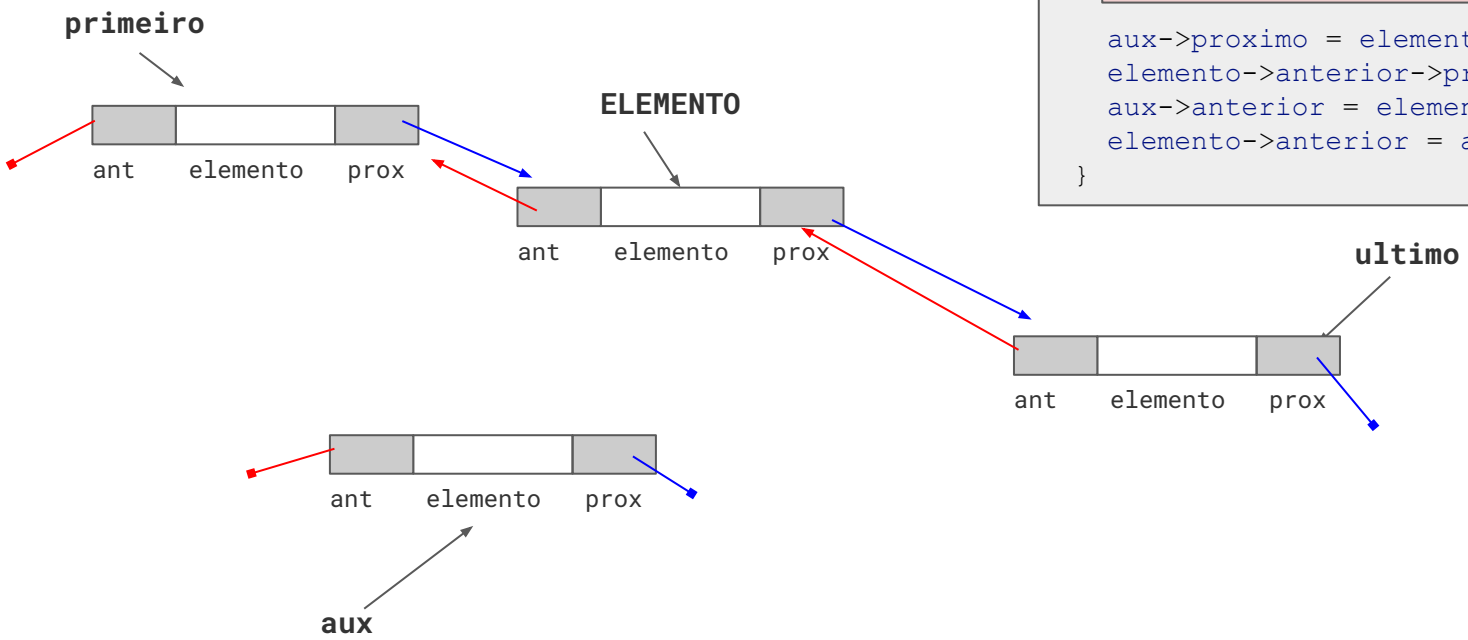


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

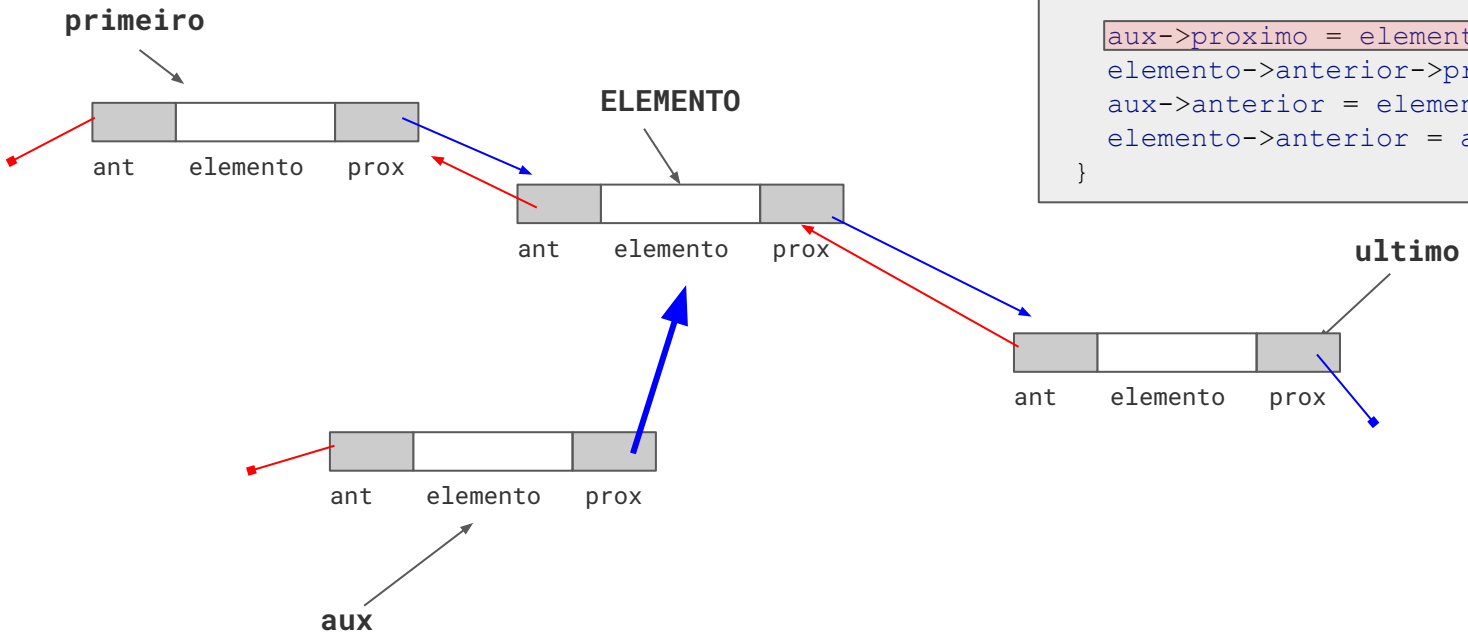


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

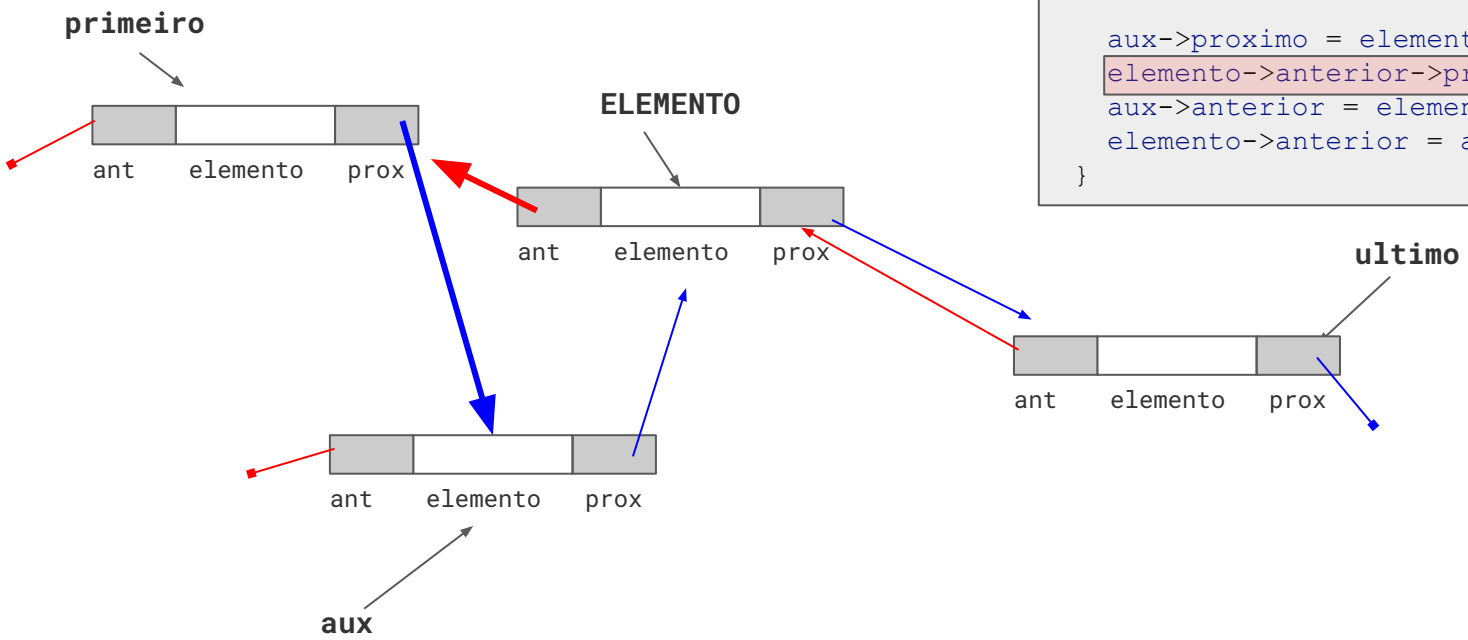


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

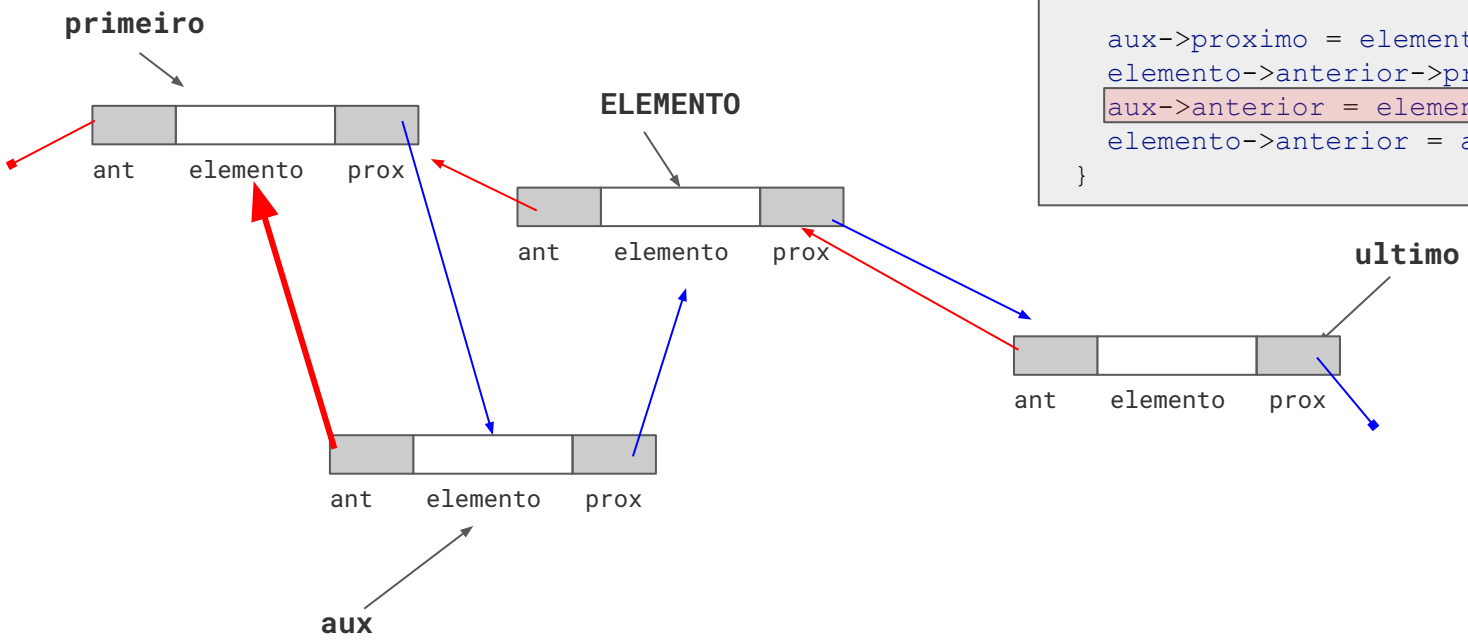


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

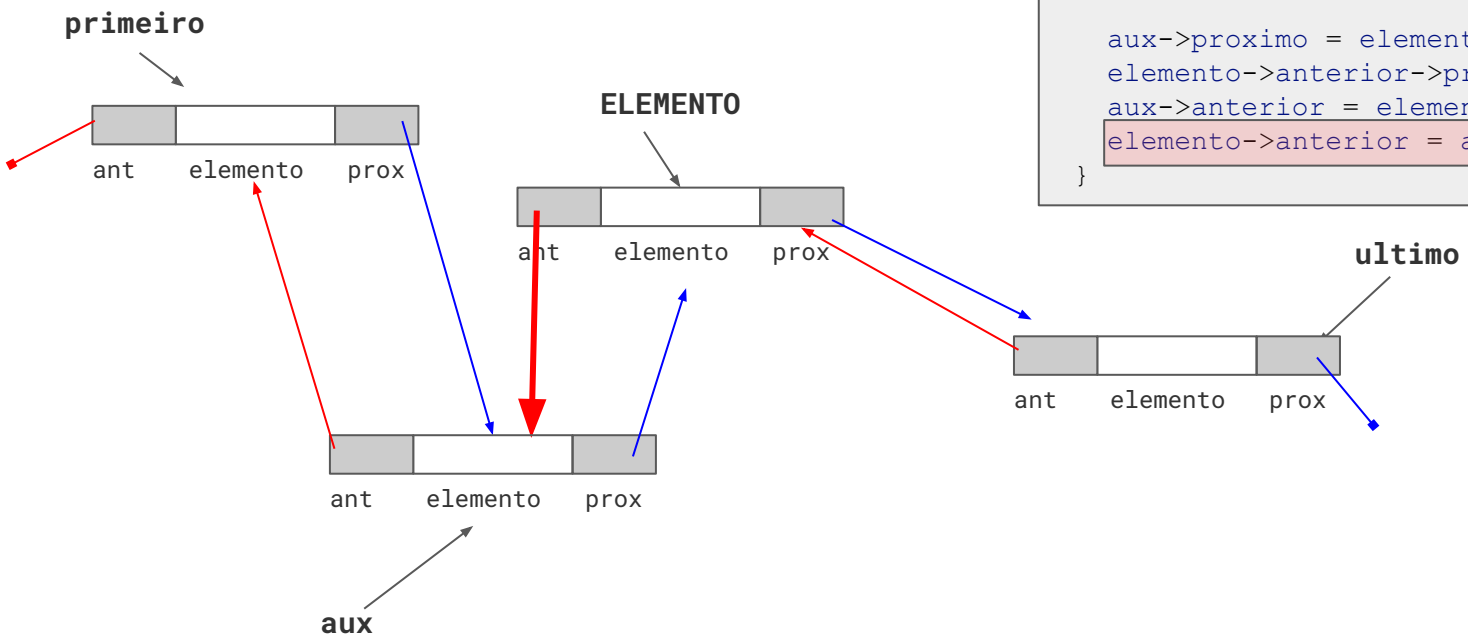


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

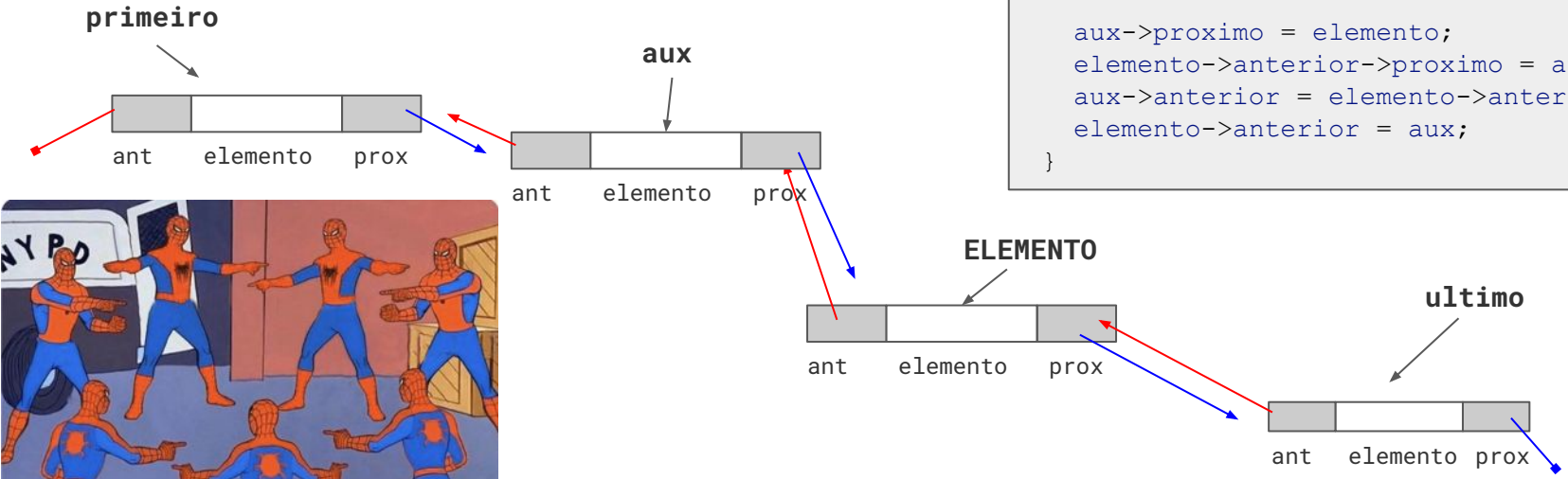


Lista Duplamente Encadeada

- Inserir novos elementos
  - **No meio**

```
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 = 20 + i;
  aux->salario = i * 1000;
  aux->proximo = NULL;
  aux->anterior = NULL;

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

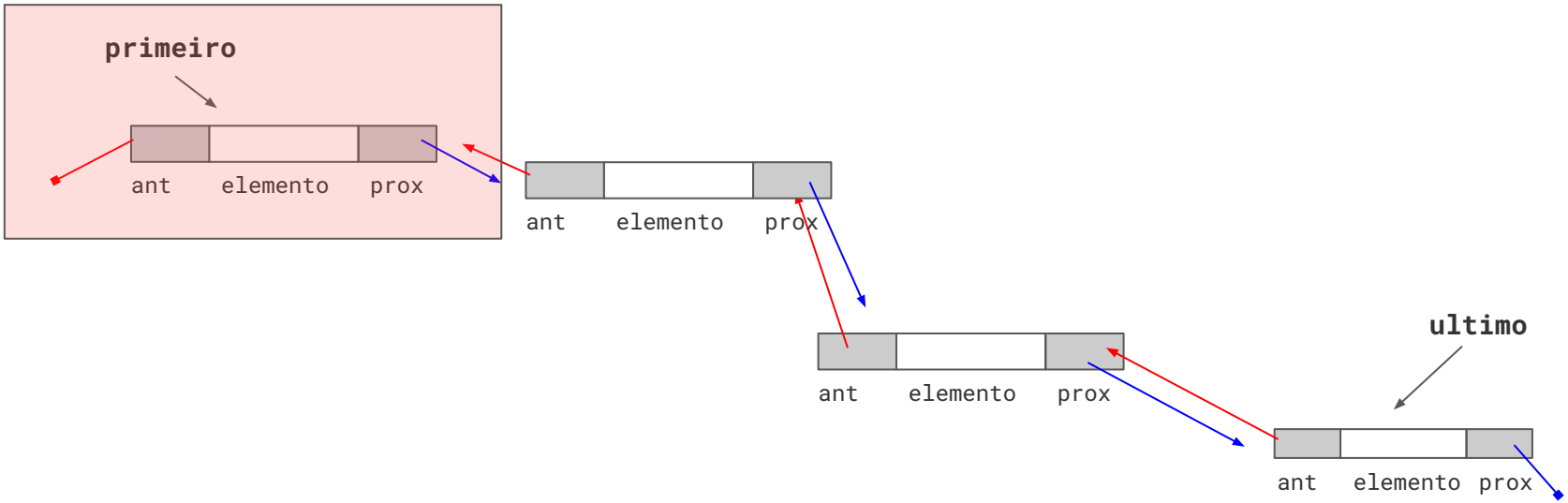


Lista Duplamente Encadeada

- Excluir um nodo
  - Excluir do início
  - Excluir do fim
  - Excluir do meio

Lista Duplamente Encadeada

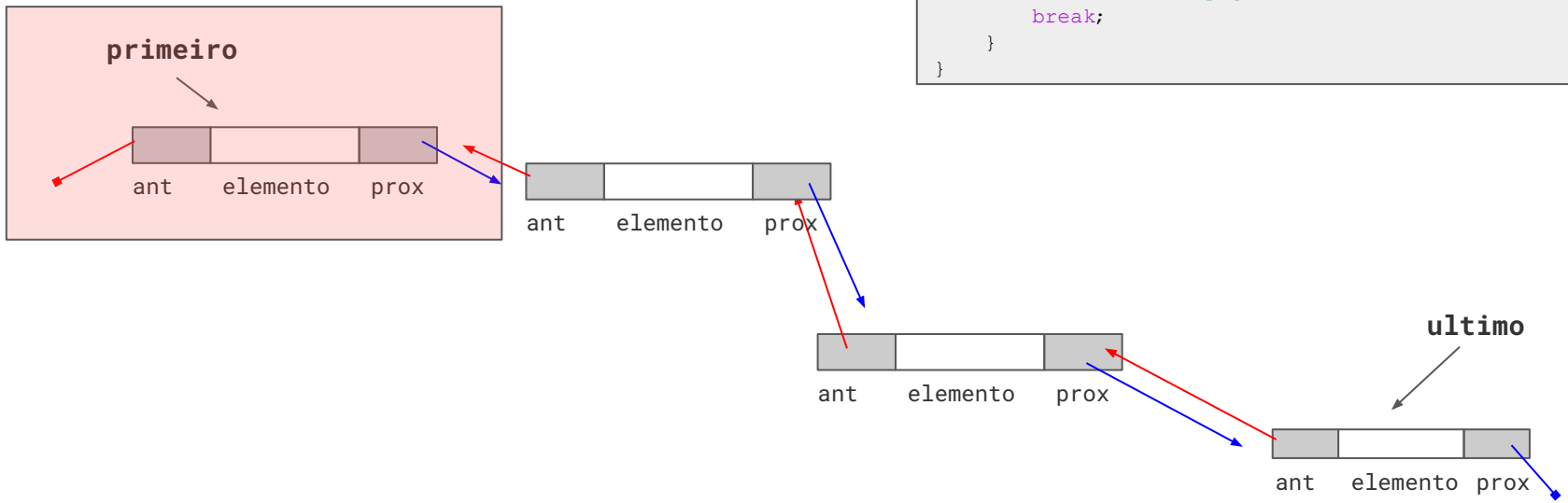
- Excluir um nodo
  - **Excluir do início**



Lista Duplamente Encadeada

- Excluir um nodo
  - **Excluir do início**

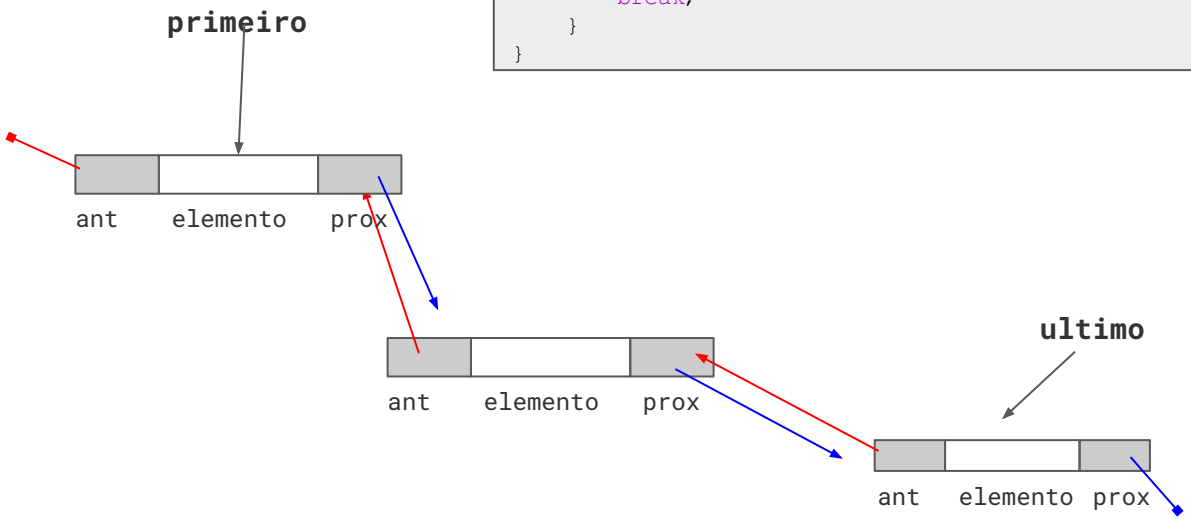
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete = 1; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
    if (aux->id == idDelete){
        if (aux == primeiro) { //verifica se é o primeiro
            primeiro = primeiro->proximo; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Lista Duplamente Encadeada

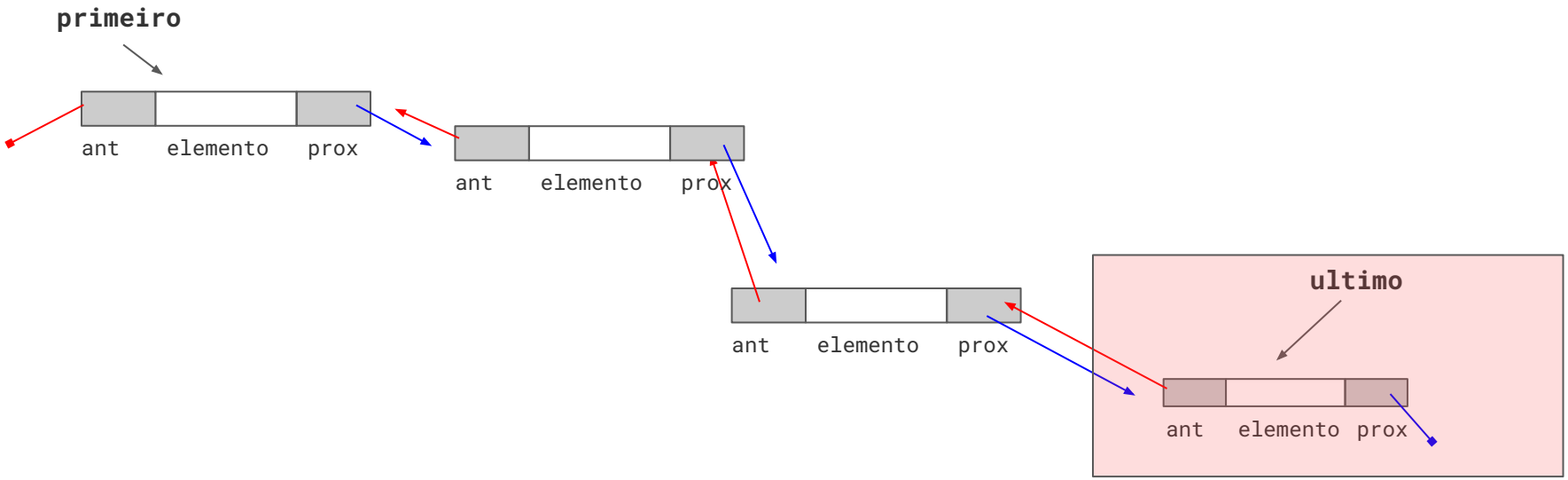
- Excluir um nodo
  - **Excluir do início**

```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
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; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Lista Duplamente Encadeada

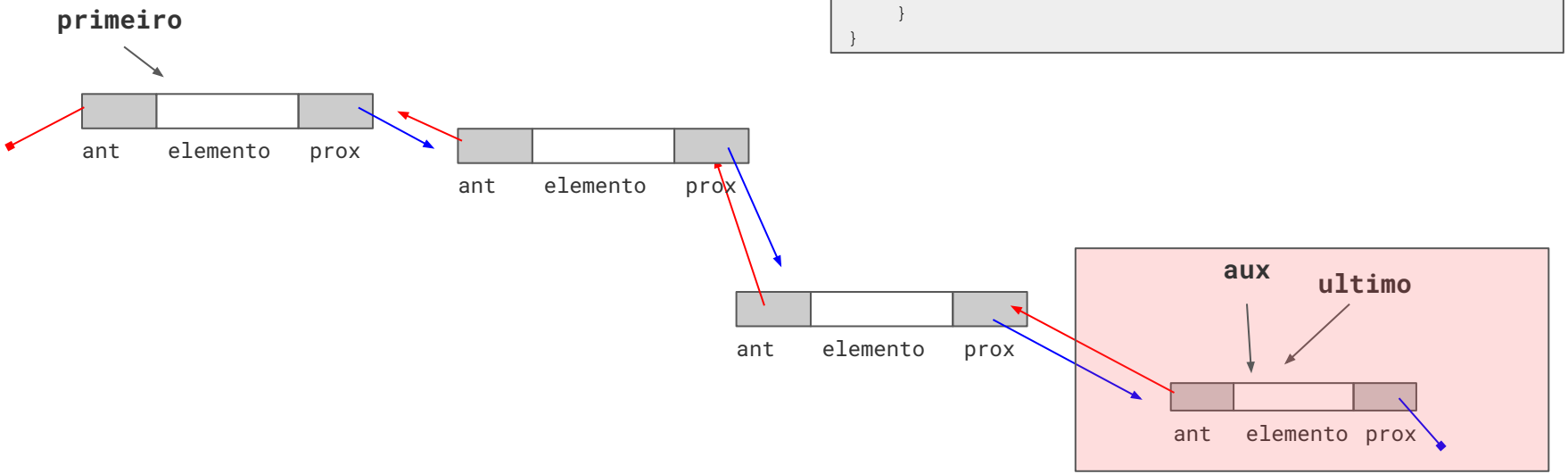
- Excluir um nodo
  - **Excluir do fim**



Lista Duplamente Encadeada

- Excluir um nodo
  - **Excluir do fim**

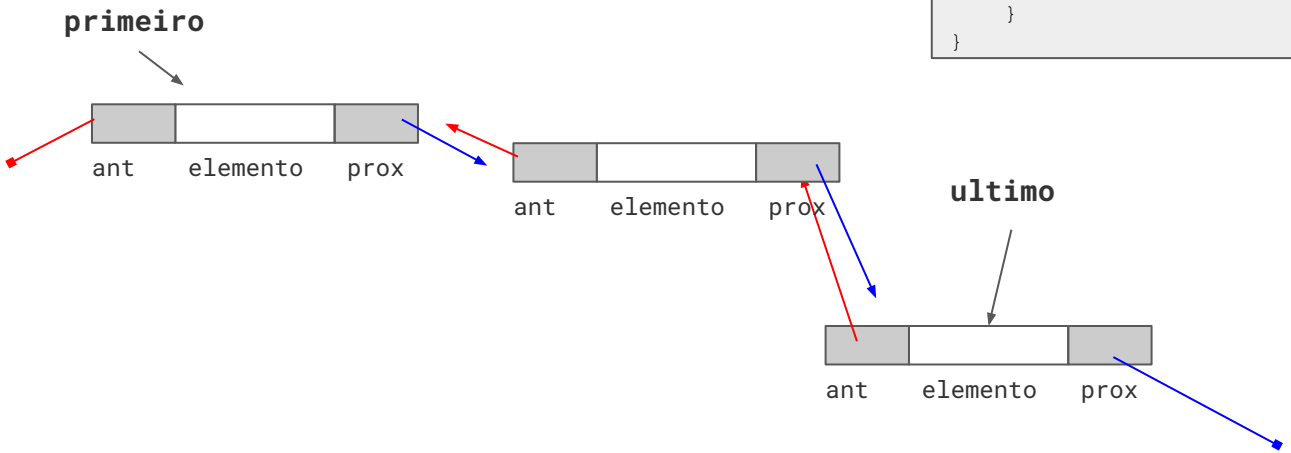
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete = 1; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
    if (aux->id == idDelete){
        if (aux == primeiro) { //verifica se é o primeiro
            primeiro = primeiro->proximo; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Lista Duplamente Encadeada

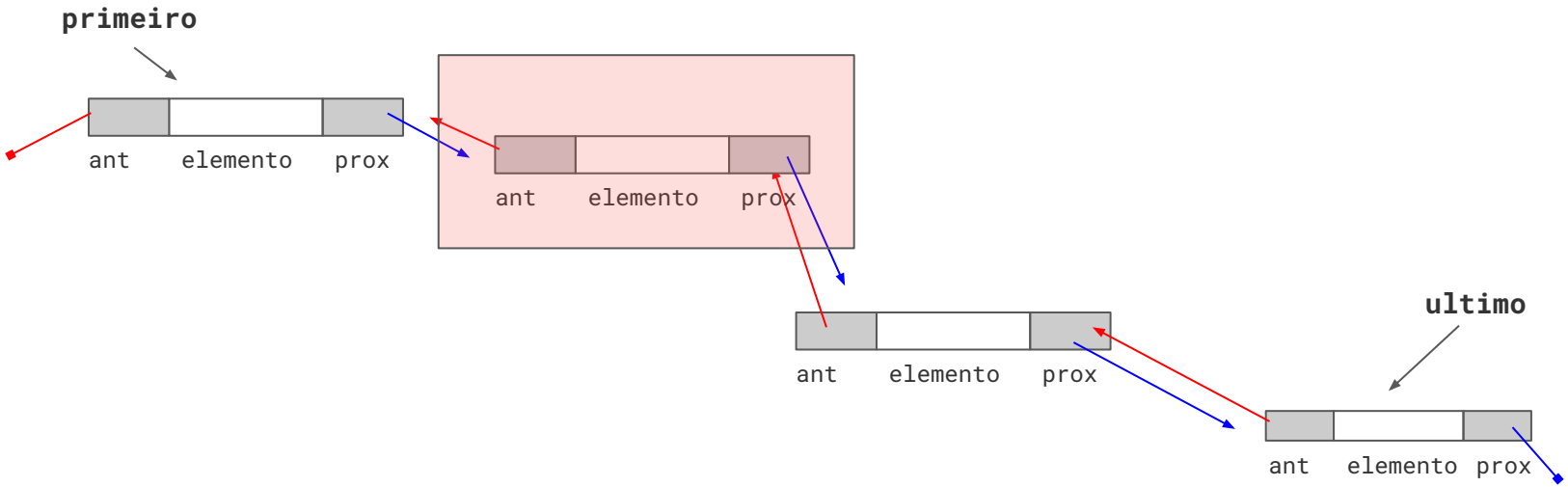
- Excluir um nodo
  - **Excluir do fim**

```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
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; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Lista Duplamente Encadeada

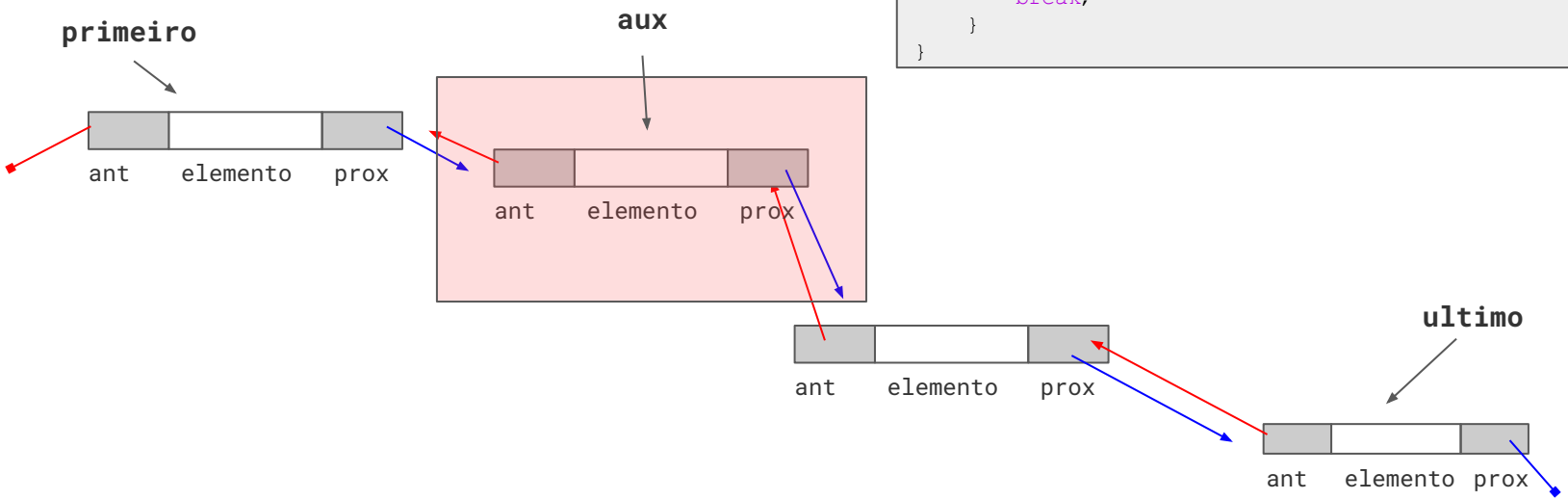
- Excluir um nodo
  - **Excluir do meio**



Lista Duplamente Encadeada

- Excluir um nodo
  - **Excluir do meio**

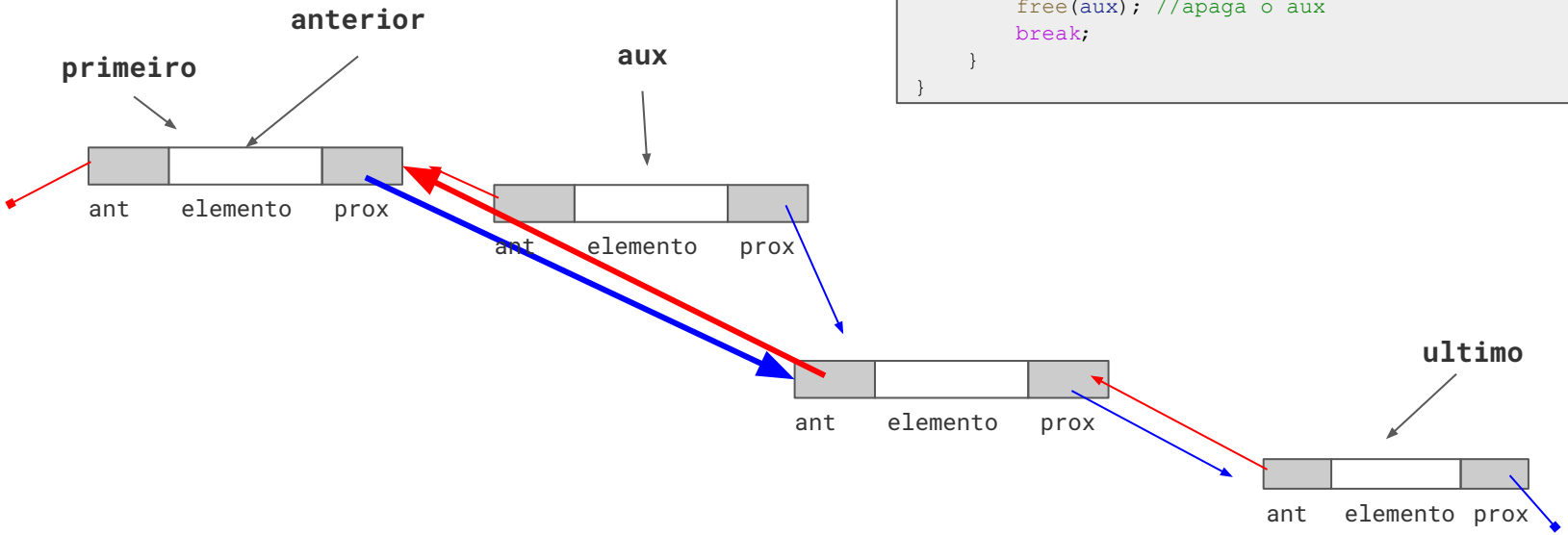
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete = 1; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
    if (aux->id == idDelete){
        if (aux == primeiro) { //verifica se é o primeiro
            primeiro = primeiro->proximo; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Lista Duplamente Encadeada

- Excluir um nodo
  - **Excluir do meio**

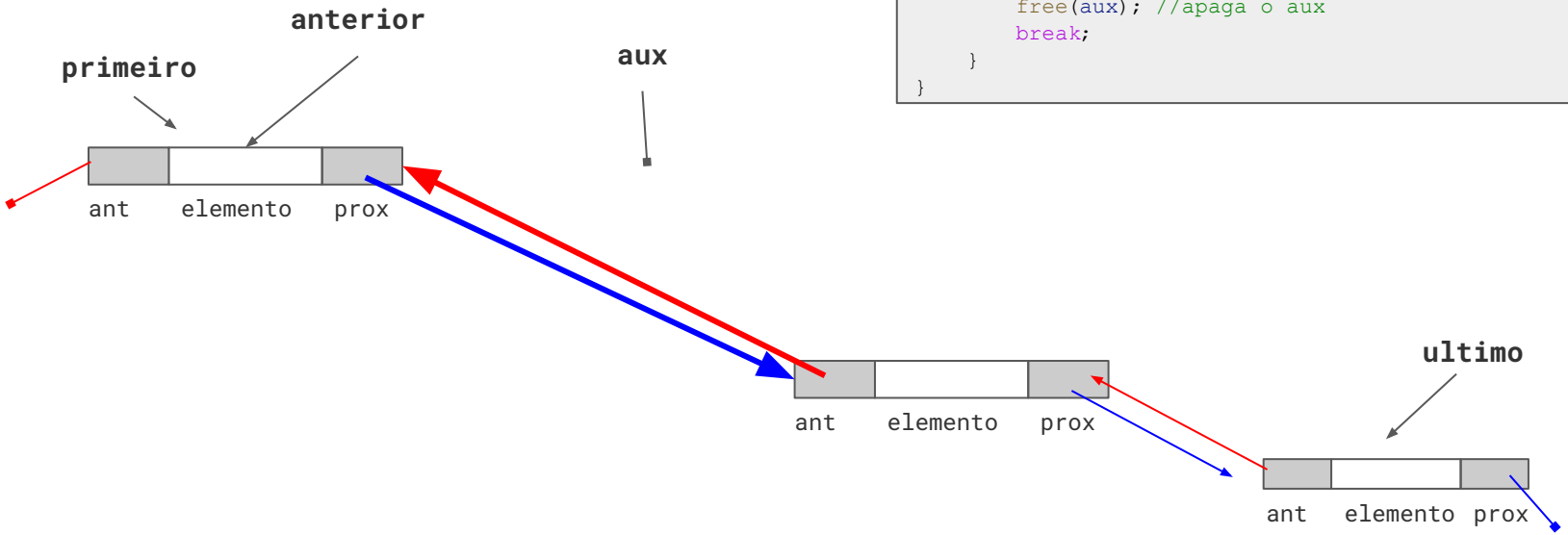
```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
int idDelete = 1; //id do funcionario a ser apagado
for (aux = primeiro; aux != NULL; aux = aux->proximo){
    if (aux->id == idDelete){
        if (aux == primeiro) { //verifica se é o primeiro
            primeiro = primeiro->proximo; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Lista Duplamente Encadeada

- Excluir um nodo
  - **Excluir do meio**

```
Funcionario *aux, *anterior; //vai ser nosso 'contador'
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; /
            primeiro->anterior = NULL;
        }
        else 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;
    }
}
```



Exercício

1. Transforme a estrutura da lista implementada no trabalho Lista 2 em uma lista duplamente encadeada.

Codeshare aula 27/10:

https://codeshare.io/kmW984

https://codeshare.io/4evwNK