

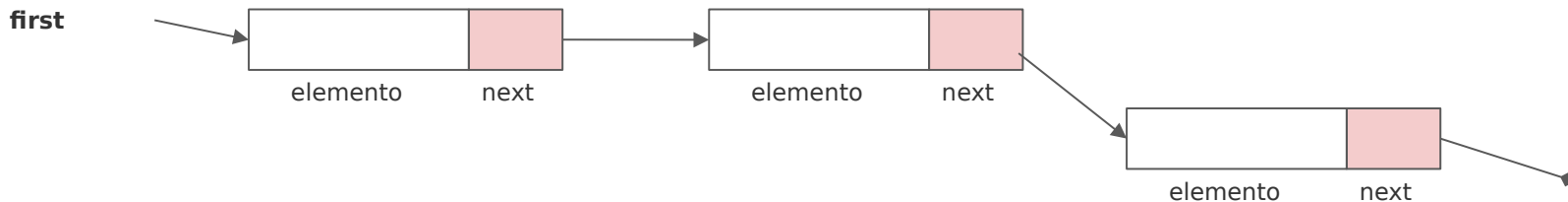
Listas Duplamente Encadeadas

Prof. Denio Duarte

Prof. Geomar Schreiner

Lista encadeada

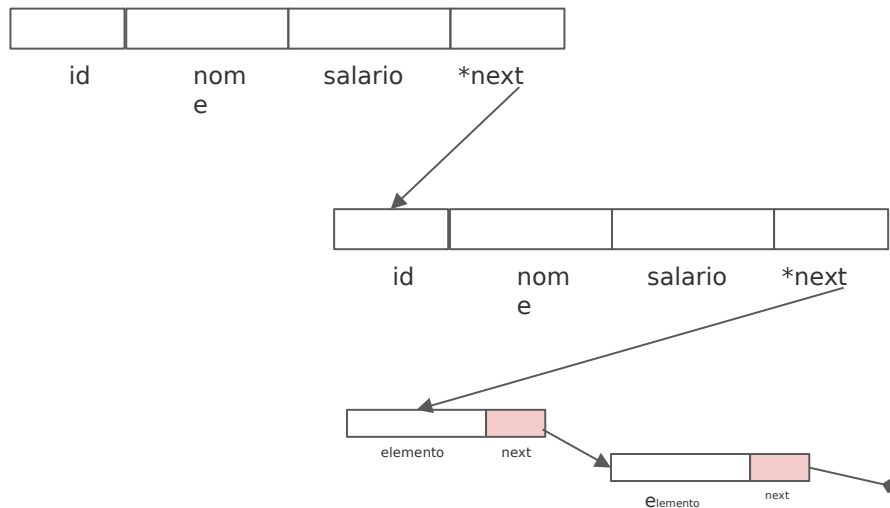
- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
 - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista



Lista encadeada simples

- 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

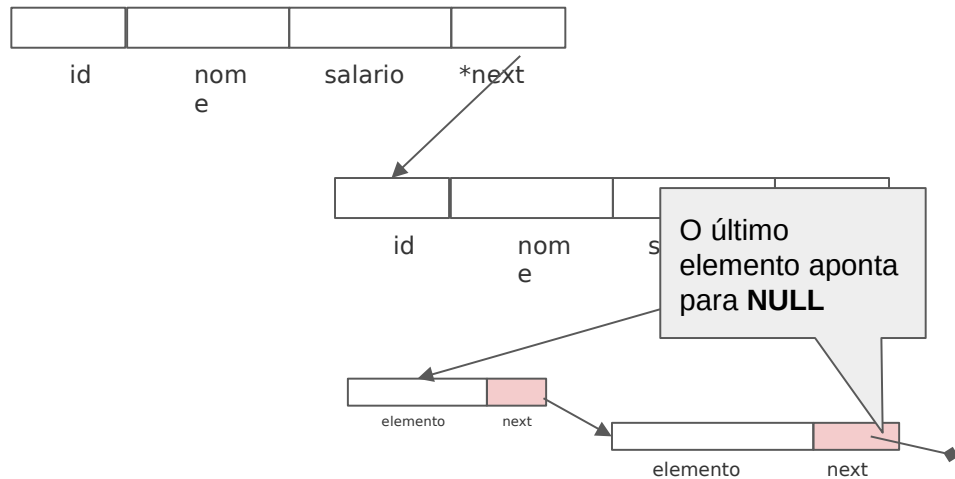
```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```



Lista encadeada simples

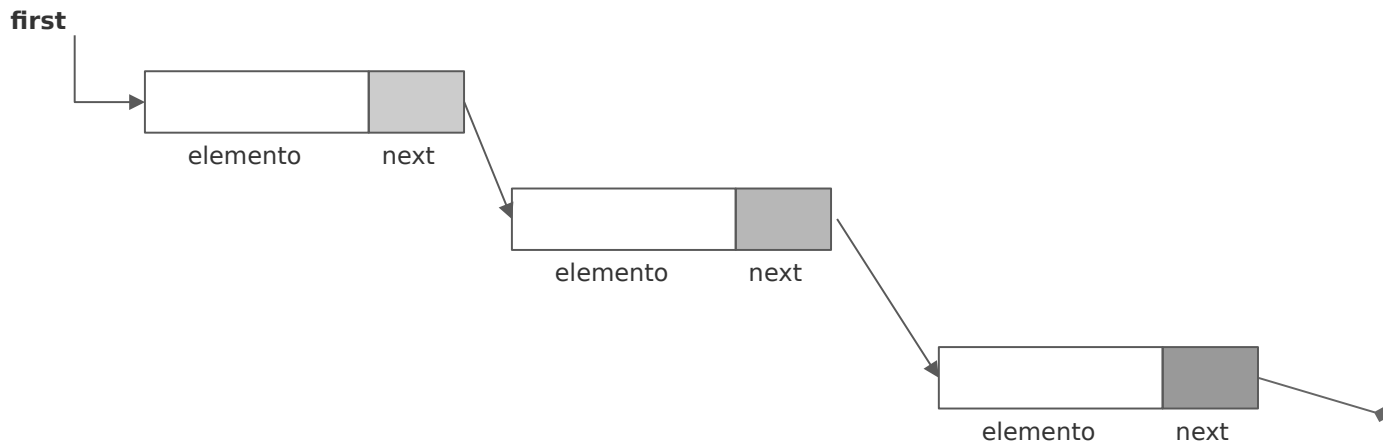
- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
 - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista
 - Sabe-se quem é o último elemento pois aponta para **NULL** (responsabilidade do programador garantir isso)

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



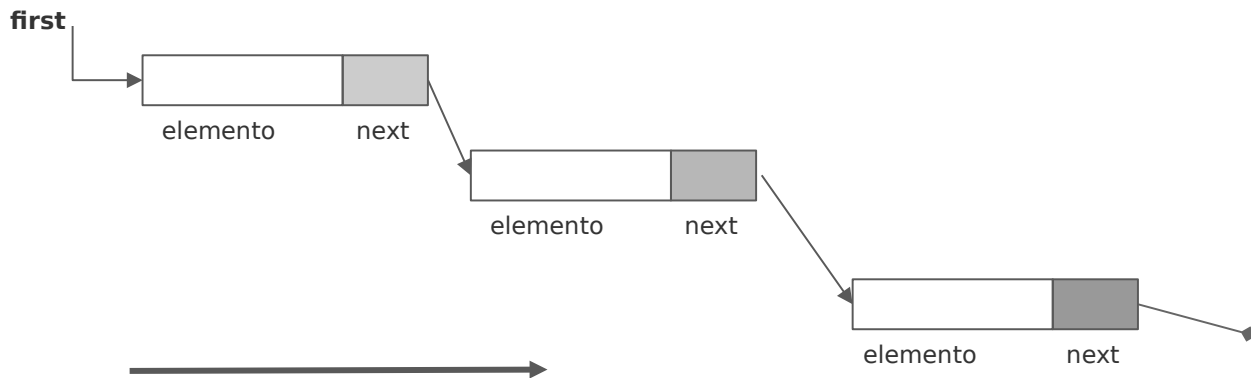
Listas

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



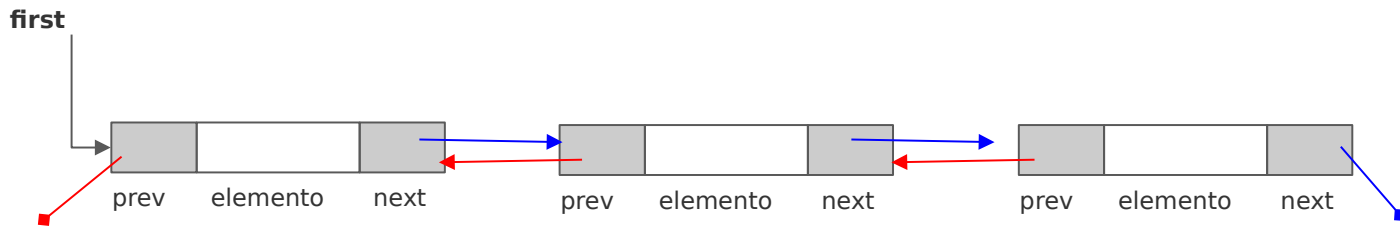
Listas

- 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.



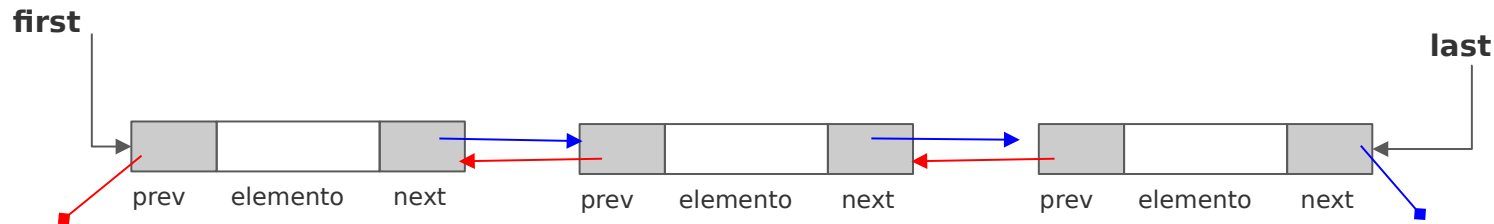
Listas Duplamente Encadeadas

- 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 Duplamente Encadeadas

- 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 first (**head**) e o last (**tail**) elemento da lista



Listas Duplamente Encadeadas

- 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 first (**head**) e o last (**tail**) elemento da lista

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```



Listas Duplamente Encadeadas

- 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 first (**head**) e o last (**tail**) elemento da lista

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

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



Listas Duplamente Encadeadas

- 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 first (**head**) e o last (**tail**) elemento da lista

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

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

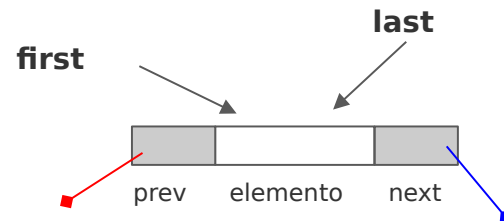


Listas Duplamente Encadeadas

- Inserir um elemento

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *first, *last;  
first = malloc (sizeof(Funcionario));  
  
first->id = 1;  
first->idade = 31;  
first->salario = 234.0;  
first->next = NULL;  
first->previous = NULL;  
  
last = first;
```

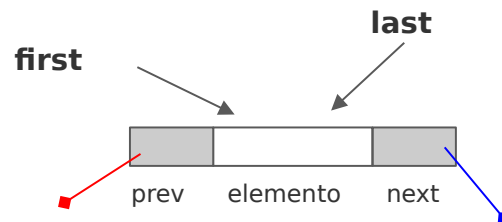


Listas Duplamente Encadeadas

- Se eu adicionar mais um, onde adiciono?
- Casos:
 - Início da lista
 - Fim da lista
 - No meio

```
Funcionario *first, *ultimo;  
first = malloc (sizeof(Funcionario));  
  
first->id = 1;  
first->idade = 31;  
first->salario = 234.0;  
first->next = NULL;  
first->previous = NULL;  
  
ultimo = first;
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

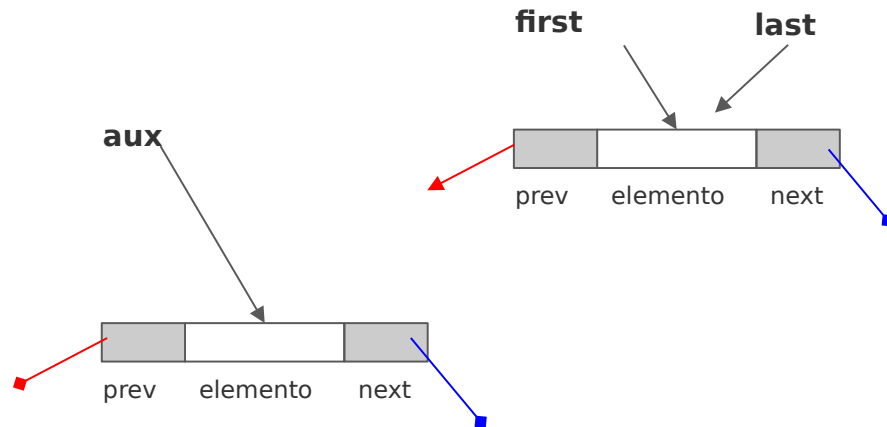


Listas Duplamente Encadeadas

- Se eu adicionar mais um, onde adiciono?
- Casos:
 - Início da lista
 - Fim da lista
 - No meio

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux;  
aux = malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->previous = NULL;  
if (elemento antes do first) {  
    aux->next = first;  
    first->previous = aux;  
    first = aux;  
}
```

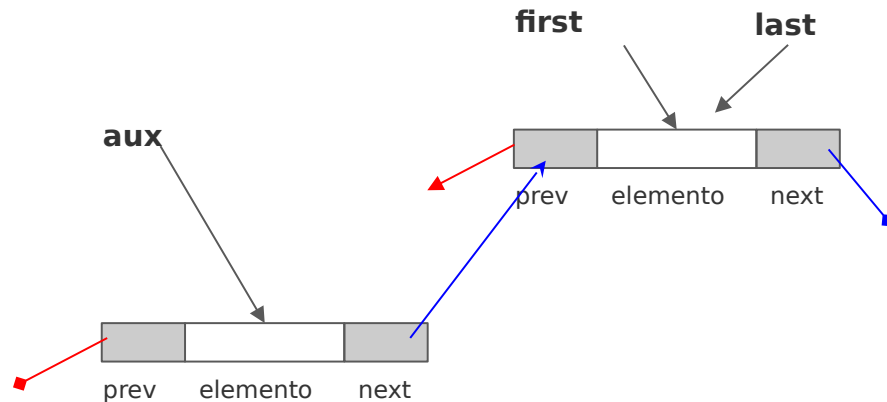


Listas Duplamente Encadeadas

- Se eu adicionar mais um, onde adiciono?
- Casos:
 - Início da lista
 - Fim da lista
 - No meio

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux;  
aux = malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->previous = NULL;  
if (elemento antes do first) {  
    aux->next = first;  
    first->previous = aux;  
    first = aux;  
}
```

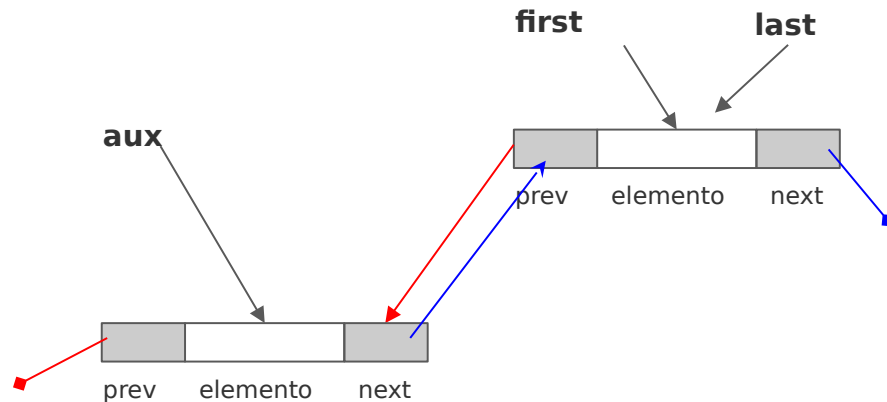


Listas Duplamente Encadeadas

- Se eu adicionar mais um, onde adiciono?
- Casos:
 - Início da lista
 - Fim da lista
 - No meio

```
Funcionario *aux;  
aux = malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->previous = NULL;  
if (elemento antes do first) {  
    aux->next = first;  
    first->previous = aux;  
    first = aux;  
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

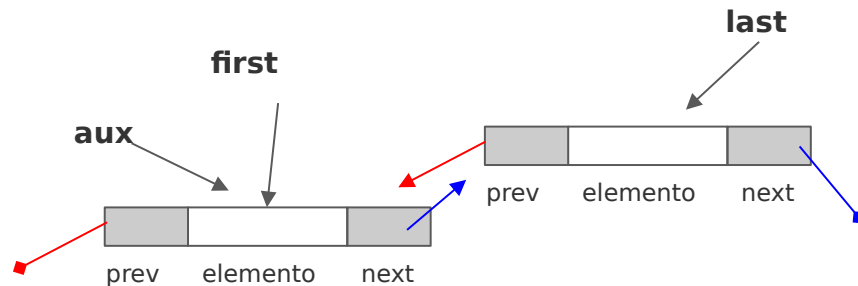


Listas Duplamente Encadeadas

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

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux;  
aux = malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->previous = NULL;  
if (elemento antes do first) {  
    aux->next = first;  
    first->previous = aux;  
    first = aux;  
}
```



Listas Duplamente Encadeadas

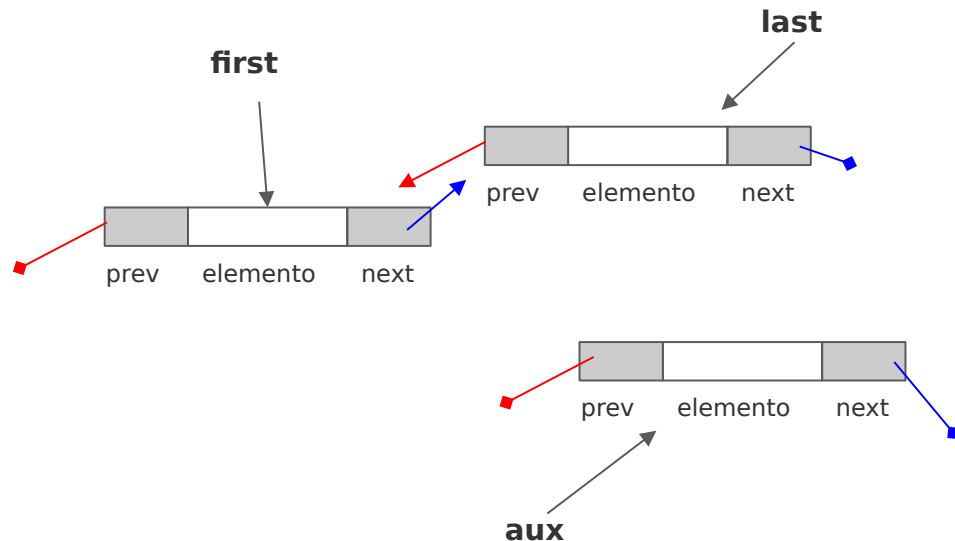
- Se eu adicionar mais um, onde adiciono
 - Início da lista
 - **Fim da lista**
 - No meio

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

```
    aux->previous = last;  
    last->next = aux;  
    last = aux;
```

```
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```



Listas Duplamente Encadeadas

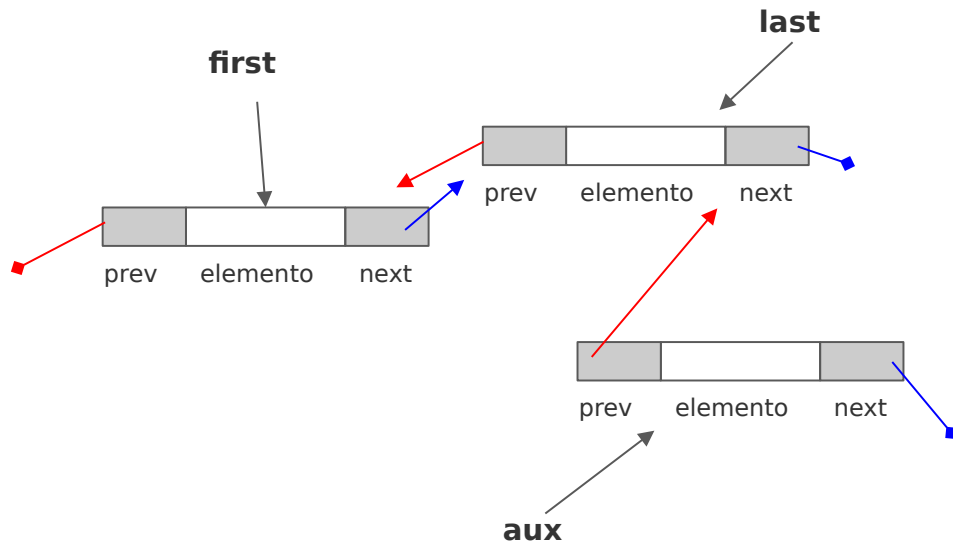
- Se eu adicionar mais um, onde adiciono
 - Início da lista
 - **Fim da lista**
 - No meio

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

```
    aux->previous = last;  
    last->next = aux;  
    last = aux;
```

```
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```



Listas Duplamente Encadeadas

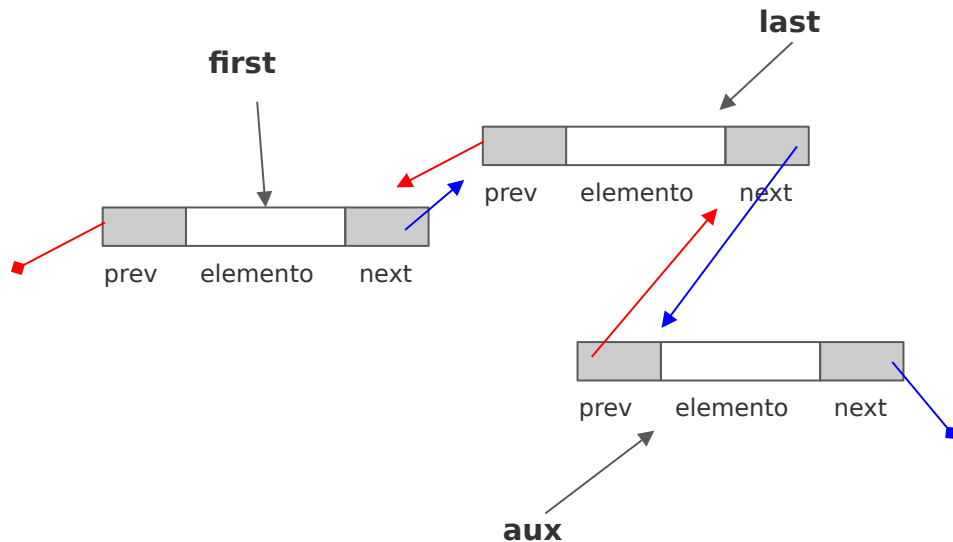
- Se eu adicionar mais um, onde adiciono
 - Início da lista
 - **Fim da lista**
 - No meio

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

```
    aux->previous = last;  
    last->next = aux;  
    last = aux;
```

```
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```



Listas Duplamente Encadeadas

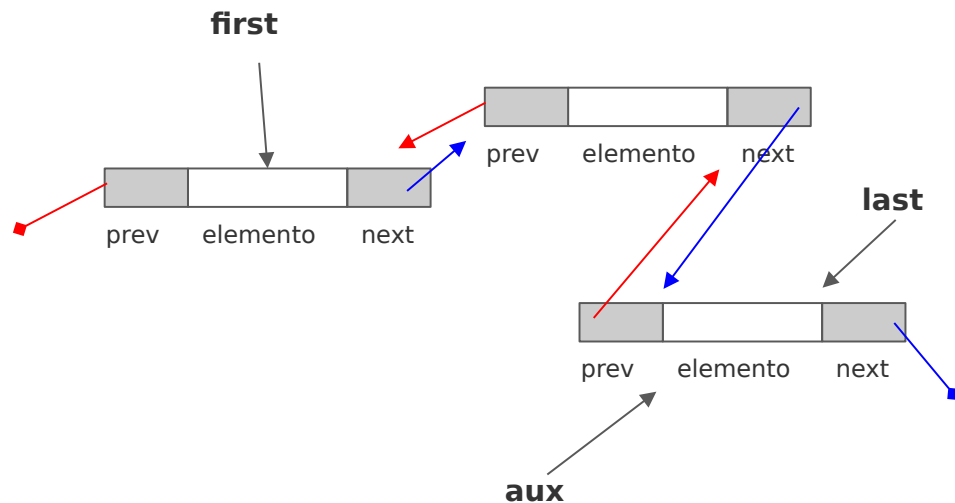
- Se eu adicionar mais um, onde adiciono
 - Início da lista
 - **Fim da lista**
 - No meio

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

```
    aux->previous = last;  
    last->next = aux;  
    last = aux;
```

```
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

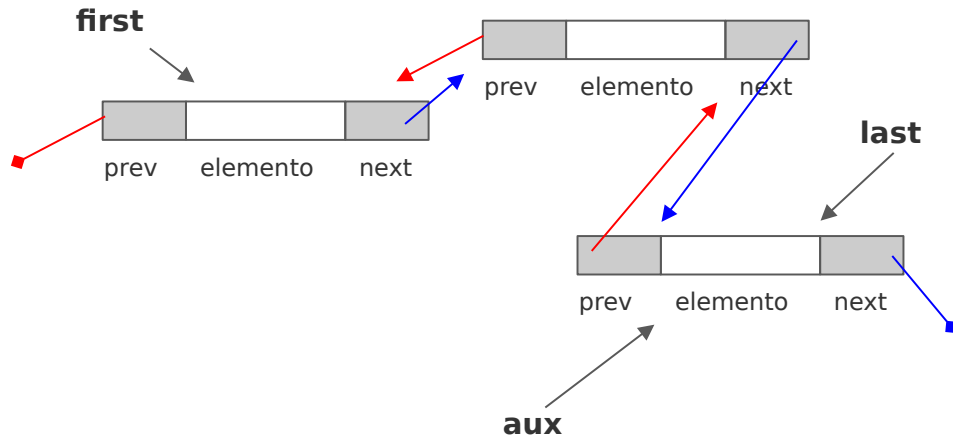


Listas Duplamente Encadeadas

- Se eu adicionar mais um, onde adiciono
 - Início da lista
 - Fim da lista
 - **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 = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->previous = NULL;  
  
    aux->next = elemento;  
    elemento->previous->next = aux;  
    aux->previous = elemento->previous;  
    elemento->previous = aux;  
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

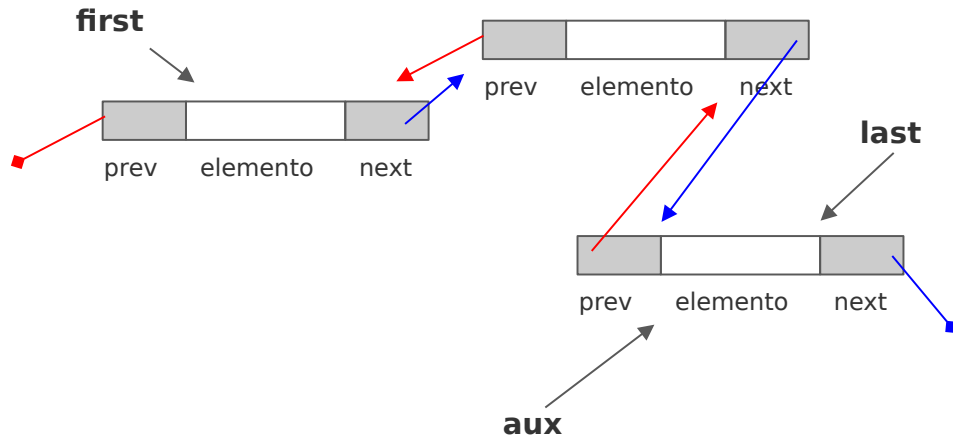


Listas Duplamente Encadeadas

- Se eu adicionar mais um, onde adiciono
 - Início da lista
 - Fim da lista
 - **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 = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->previous = NULL;  
  
    aux->next = elemento;  
    elemento->previous->next = aux;  
    aux->previous = elemento->previous;  
    elemento->previous = aux;  
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```



Listas Duplamente Encadeadas

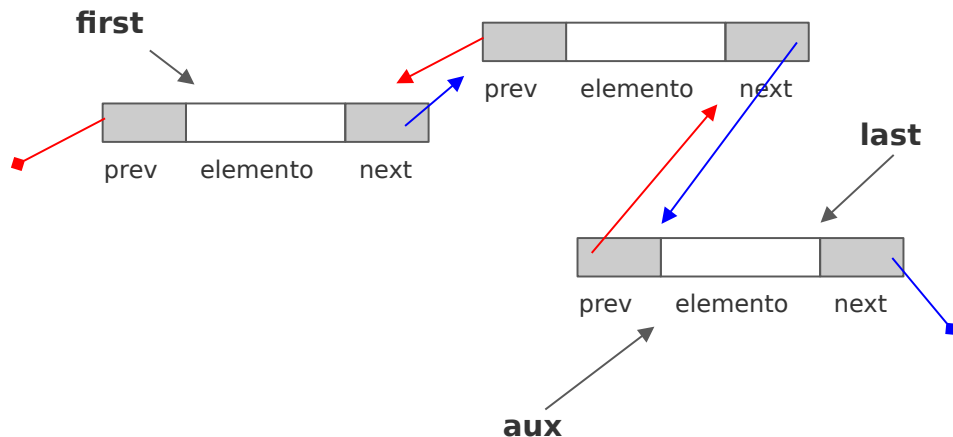
- Se eu adicionar mais um, onde adiciono

- Início
- Fim
- No

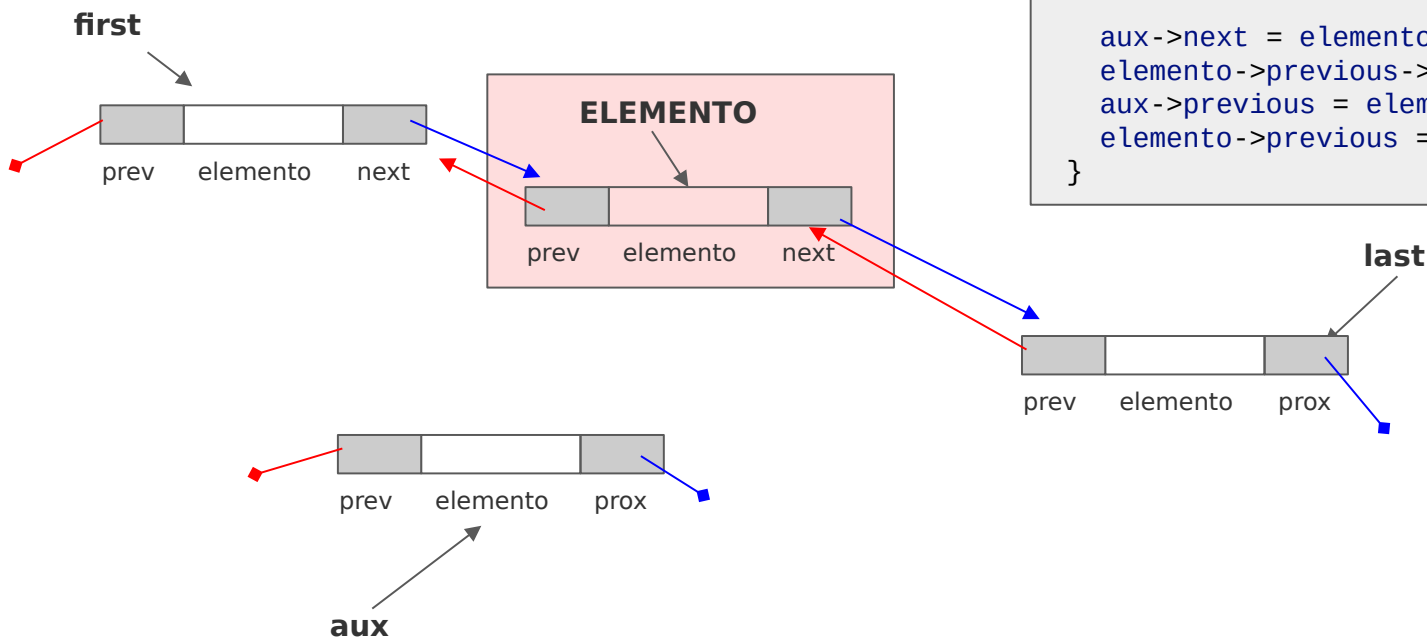
magia() não é uma função real, estamos usando para ilustrar que sabemos qual a posição correta.

```
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 = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->previous = NULL;  
  
    aux->next = elemento;  
    elemento->previous->next = aux;  
    aux->previous = elemento->previous;  
    elemento->previous = aux;  
}
```

```
struct funcionario{  
    int id;  
    int idade;  
    double salario;  
    struct funcionario *next;  
    struct funcionario *previous;  
};  
typedef struct funcionario Funcionario;
```

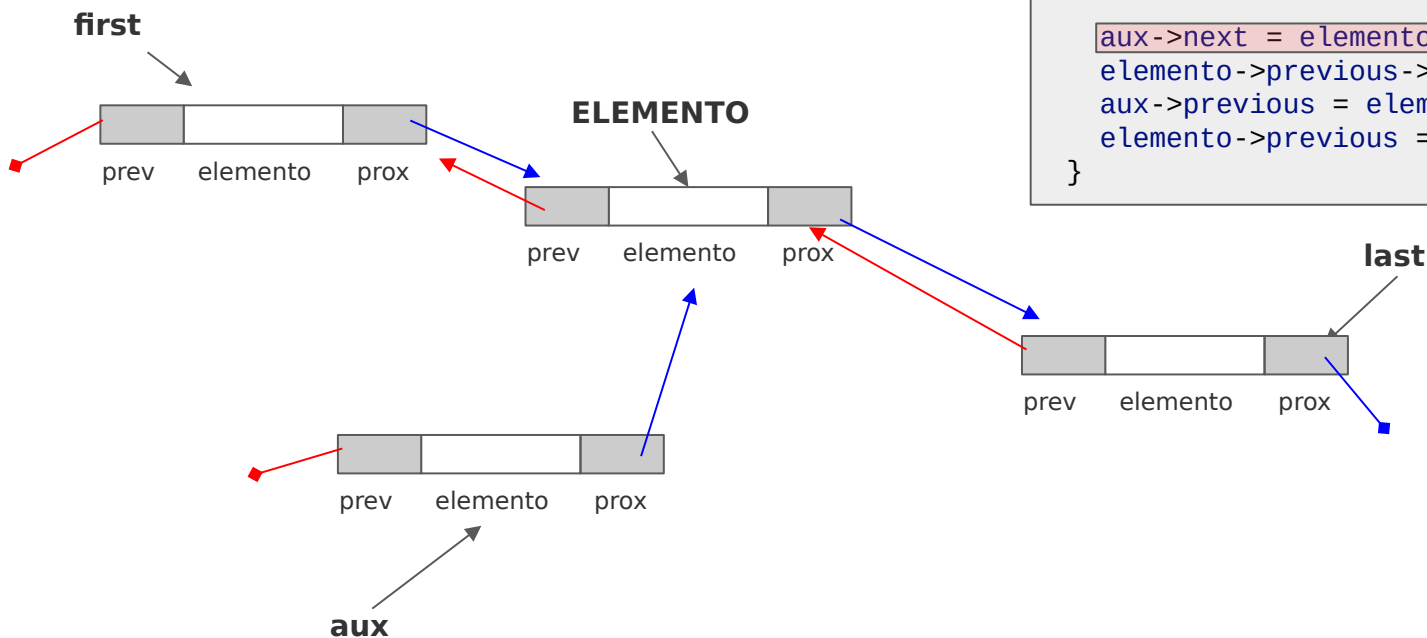


Listas Duplamente Encadeado



```
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->next = NULL;  
    aux->previous = NULL;  
  
    aux->next = elemento;  
    elemento->previous->next = aux;  
    aux->previous = elemento->previous;  
    elemento->previous = aux;  
}
```

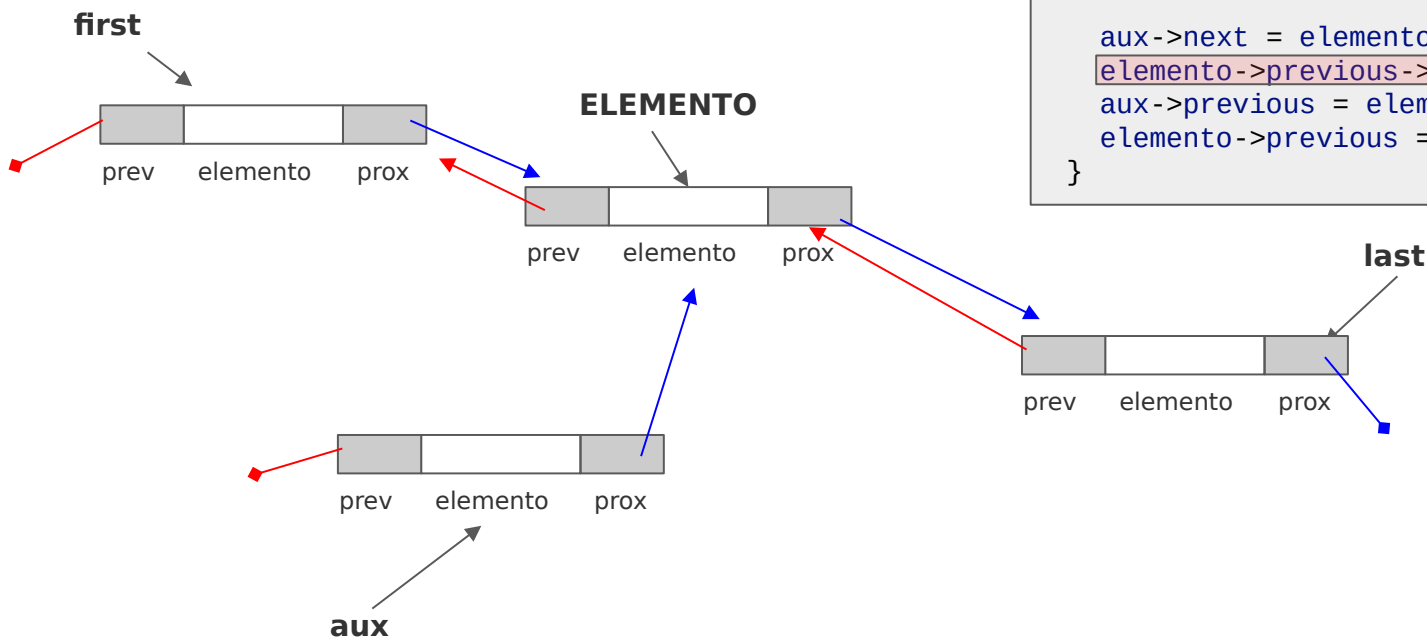
Listas



```
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->next = NULL;
    aux->previous = NULL;

    aux->next = elemento;
    elemento->previous->next = aux;
    aux->previous = elemento->previous;
    elemento->previous = aux;
}
```

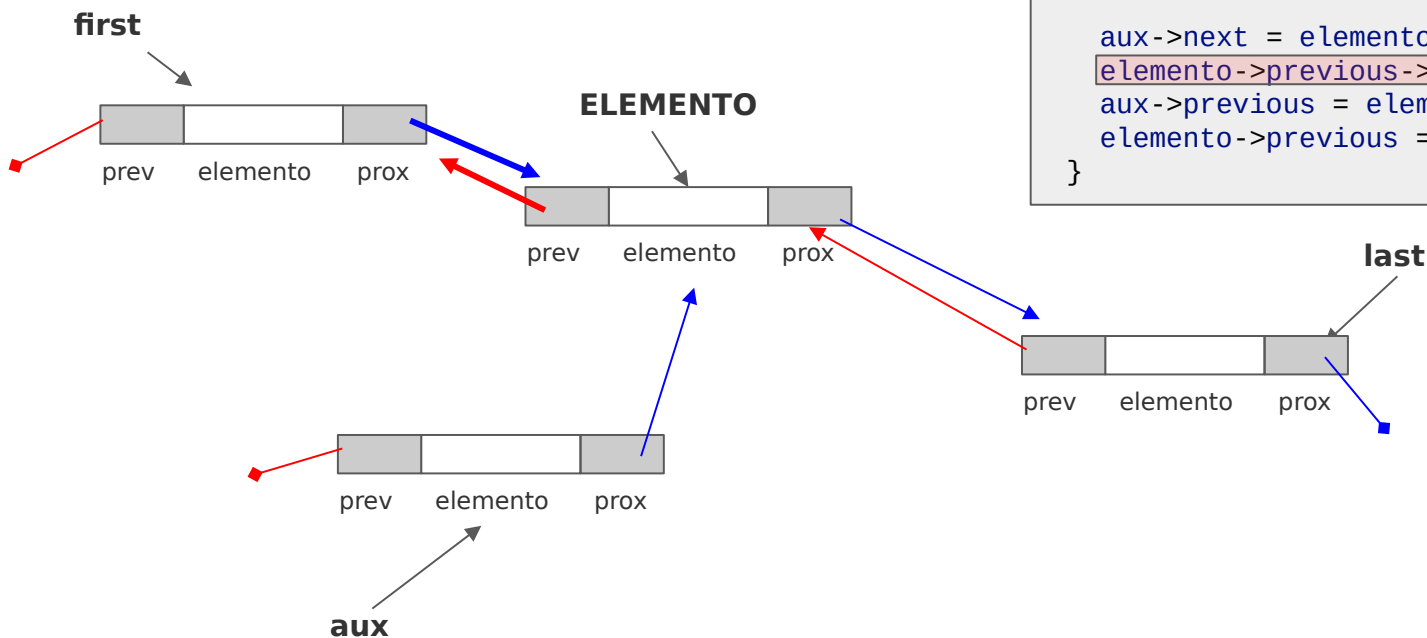
Listas



```
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->next = NULL;
    aux->previous = NULL;

    aux->next = elemento;
    elemento->previous->next = aux;
    aux->previous = elemento->previous;
    elemento->previous = aux;
}
```

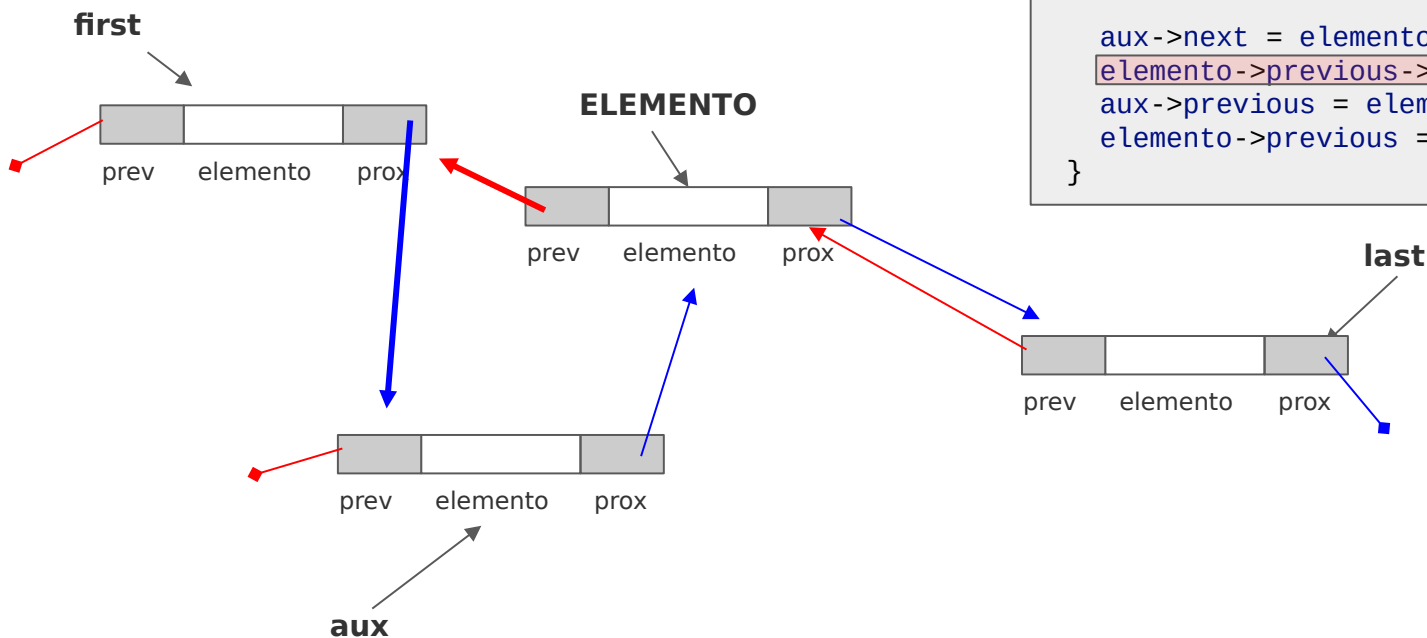
Listas



```
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->next = NULL;
    aux->previous = NULL;

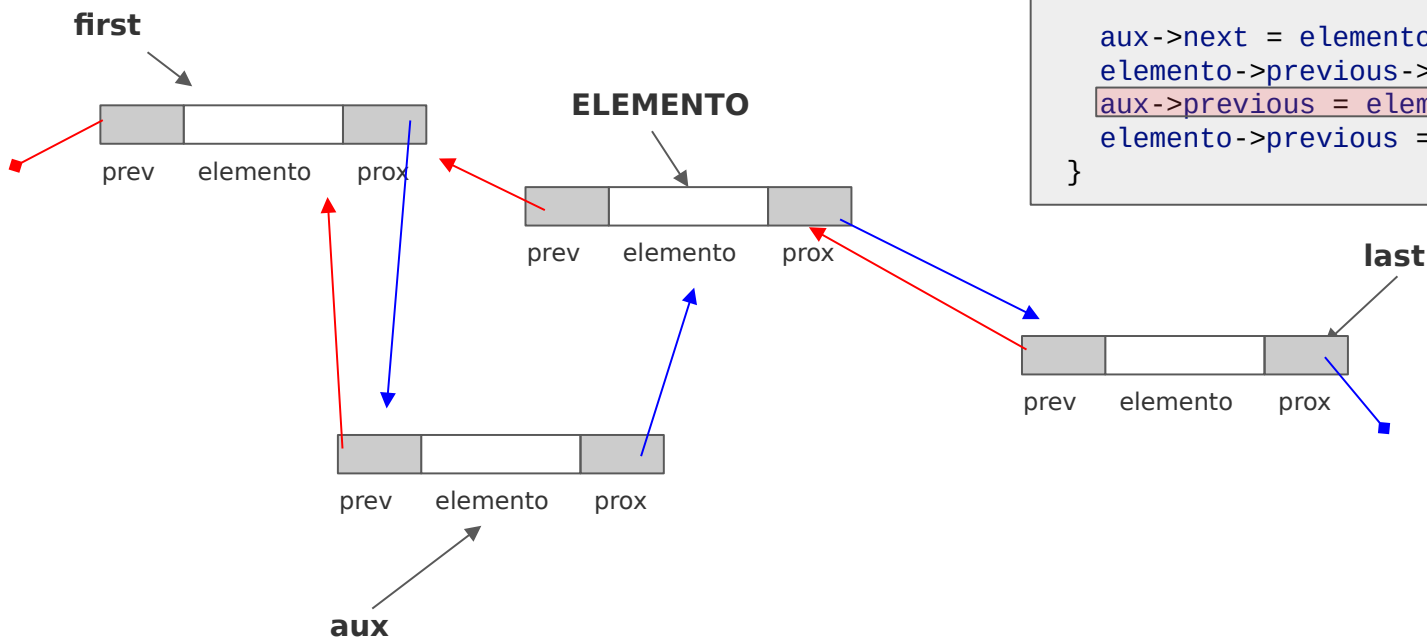
    aux->next = elemento;
    elemento->previous->next = aux;
    aux->previous = elemento->previous;
    elemento->previous = aux;
}
```

Listas



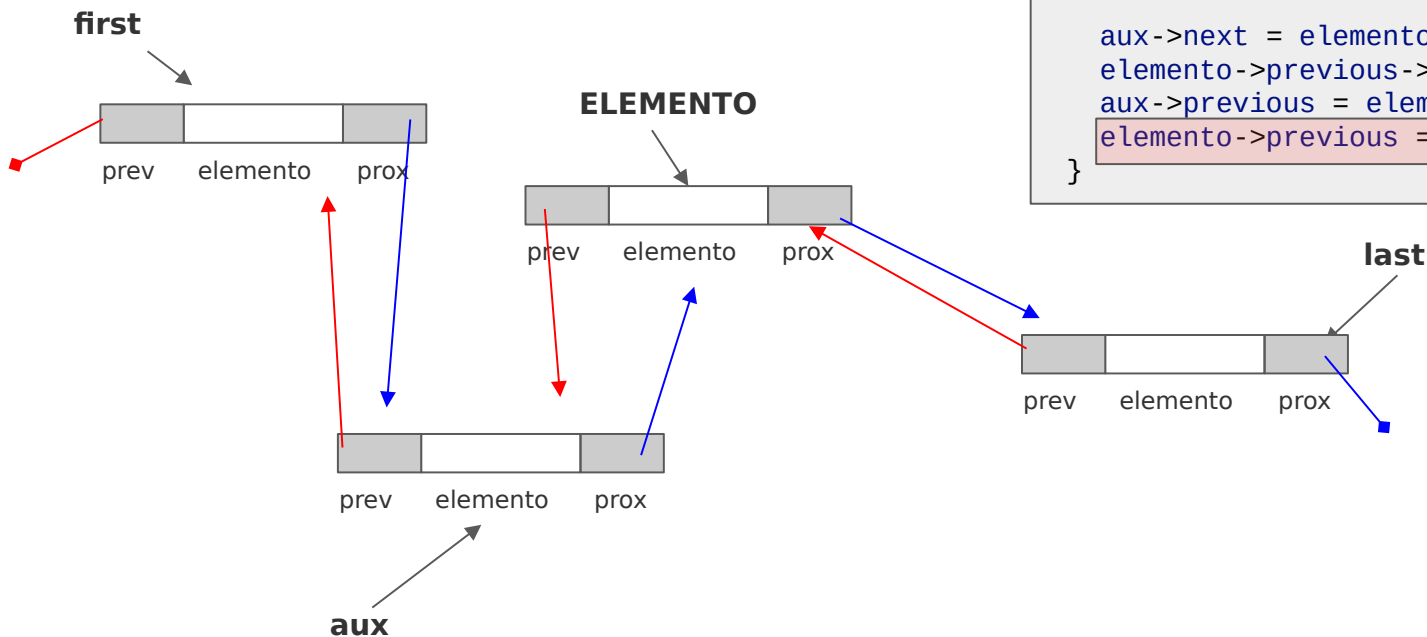
```
Funcionario *aux;  
for (i = 1; i < 10; i++){  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i+1;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->previous = NULL;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

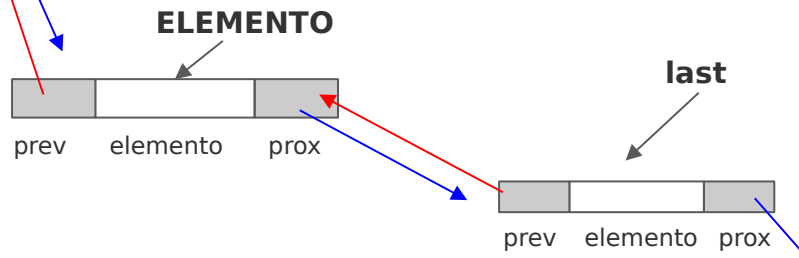
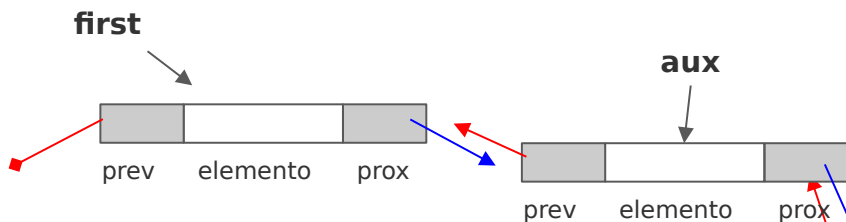
Listas



```
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->next = NULL;
    aux->previous = NULL;

    aux->next = elemento;
    elemento->previous->next = aux;
    aux->previous = elemento->previous;
    elemento->previous = aux;
}
```

Listas

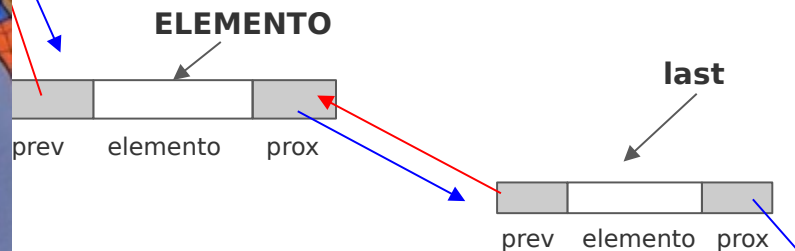


```
Funcionario *aux;  
for (i = 1; i < 10; i++){  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i+1;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->previous = NULL;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


Listas

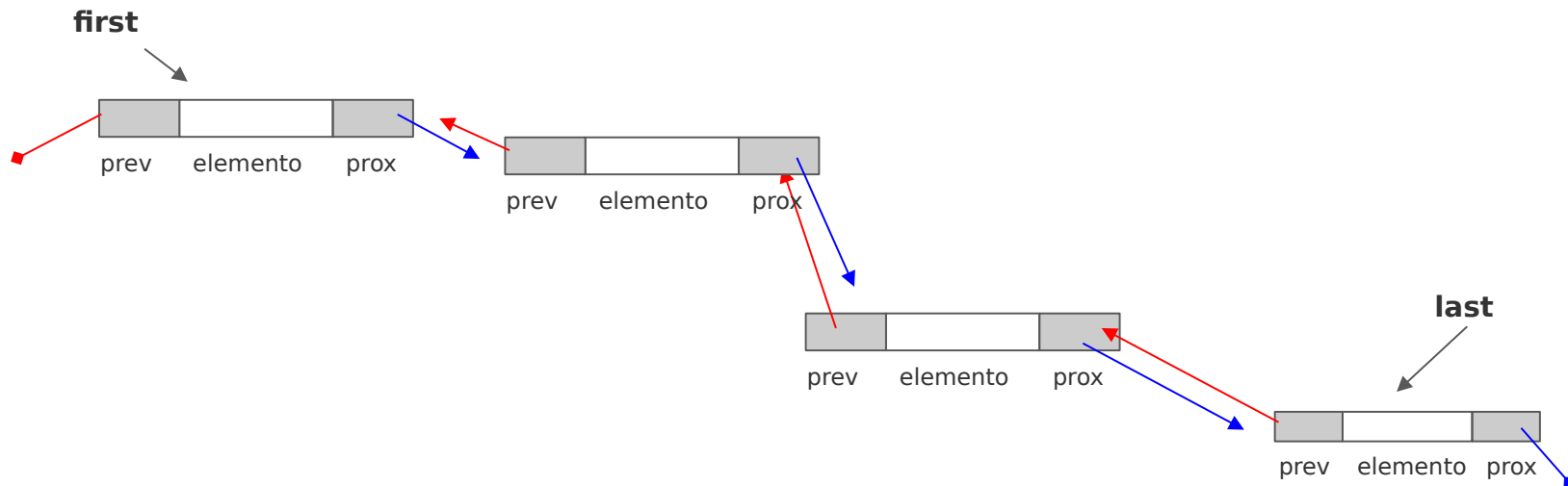


```
Funcionario *aux;  
for (i = 1; i < 10; i++){  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i+1;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->previous = NULL;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



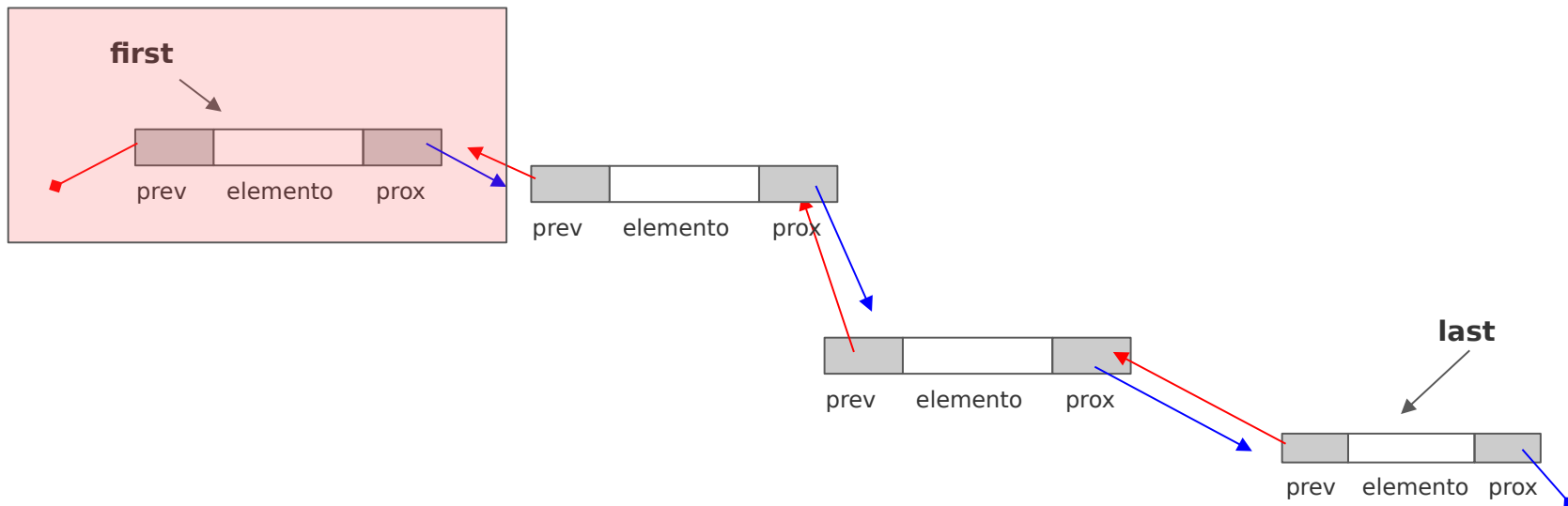
Listas

- Deletar um item



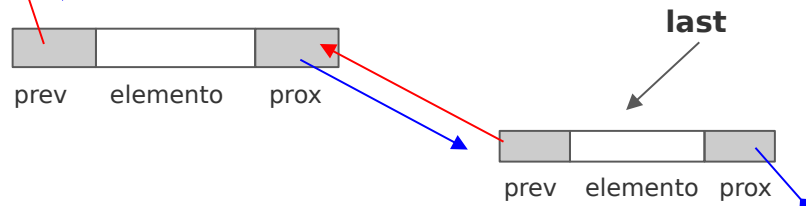
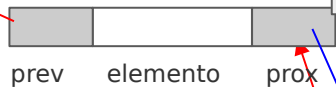
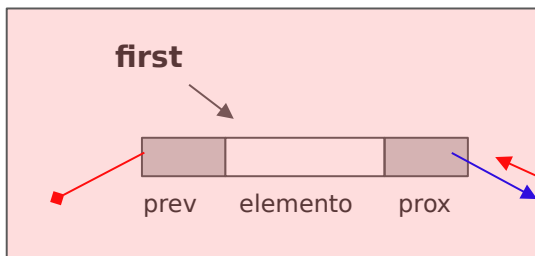
Listas

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



Listas

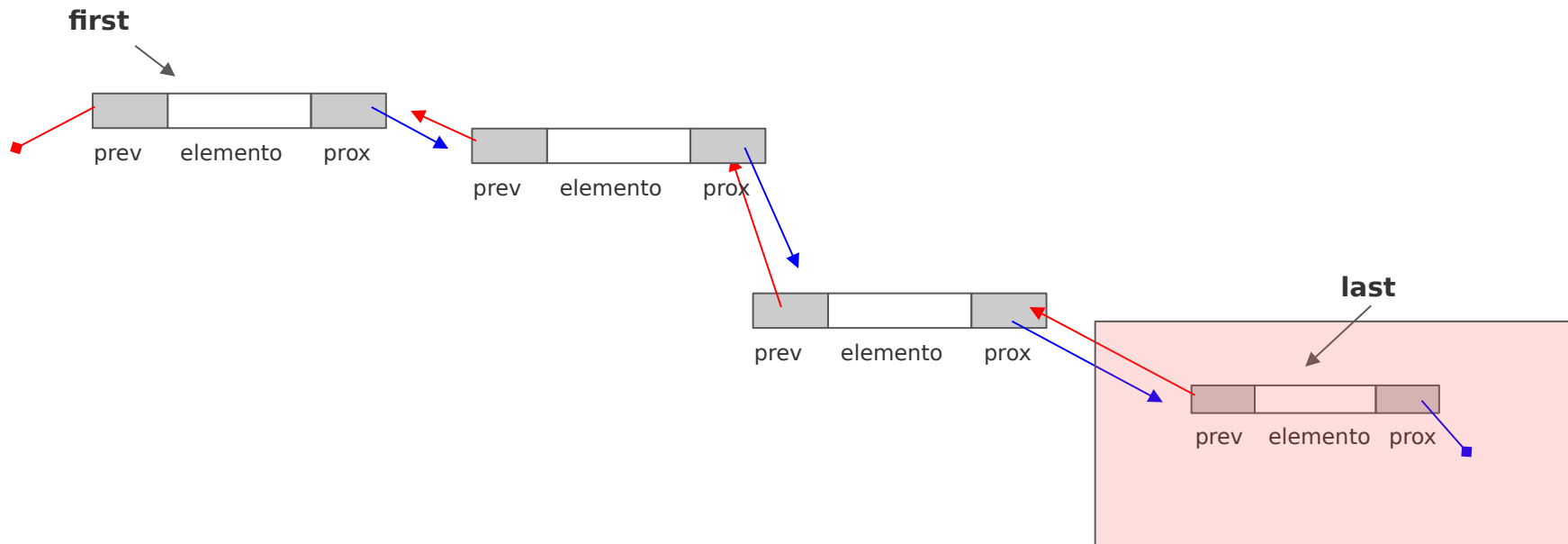
- Para eliminar um item devemos considerar:
 - Eliminar do início



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

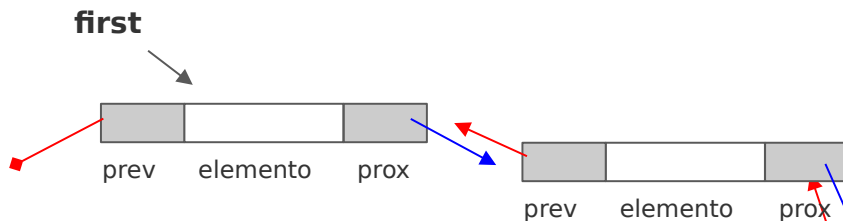
Listas

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

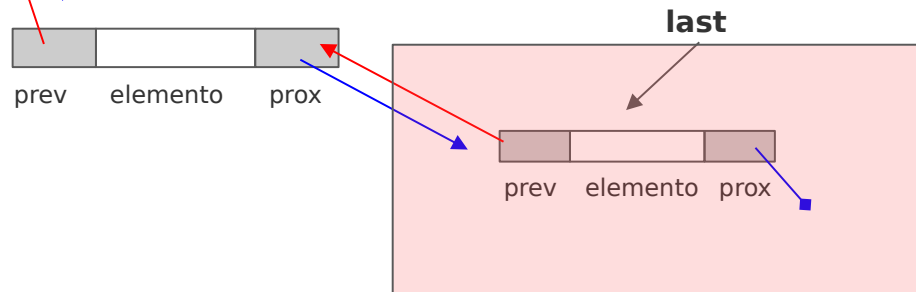


Listas

- Para eliminar um item devemos considerar:
 - Eliminar do início
 - Eliminar do fim

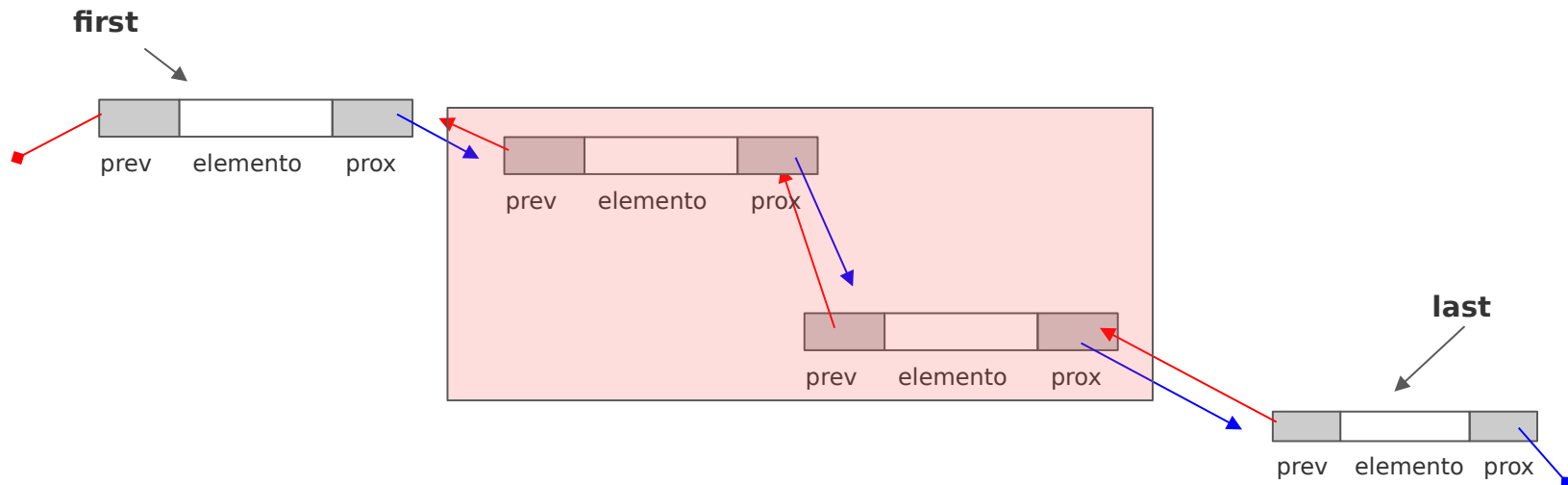


```
Funcionario *aux, *previous; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = first; aux != NULL; aux = aux->prox){
    if (aux->id == idDelete){
        if (aux == first) { //verifica se é o first
            first = first->next; /
            first->previous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



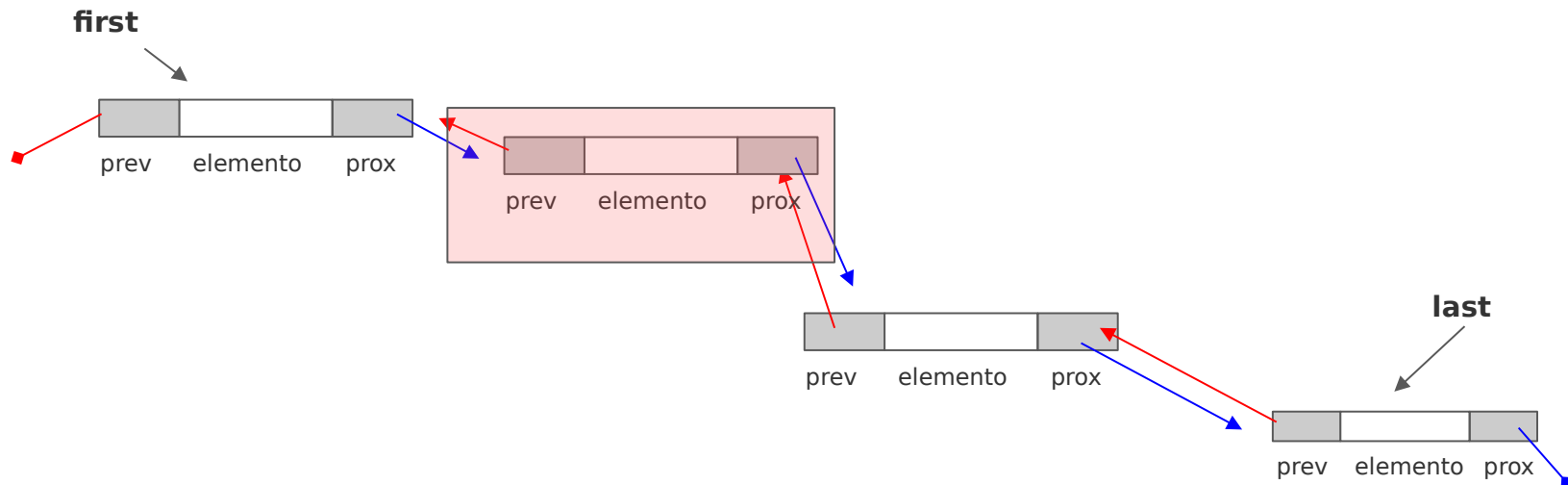
Listas

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



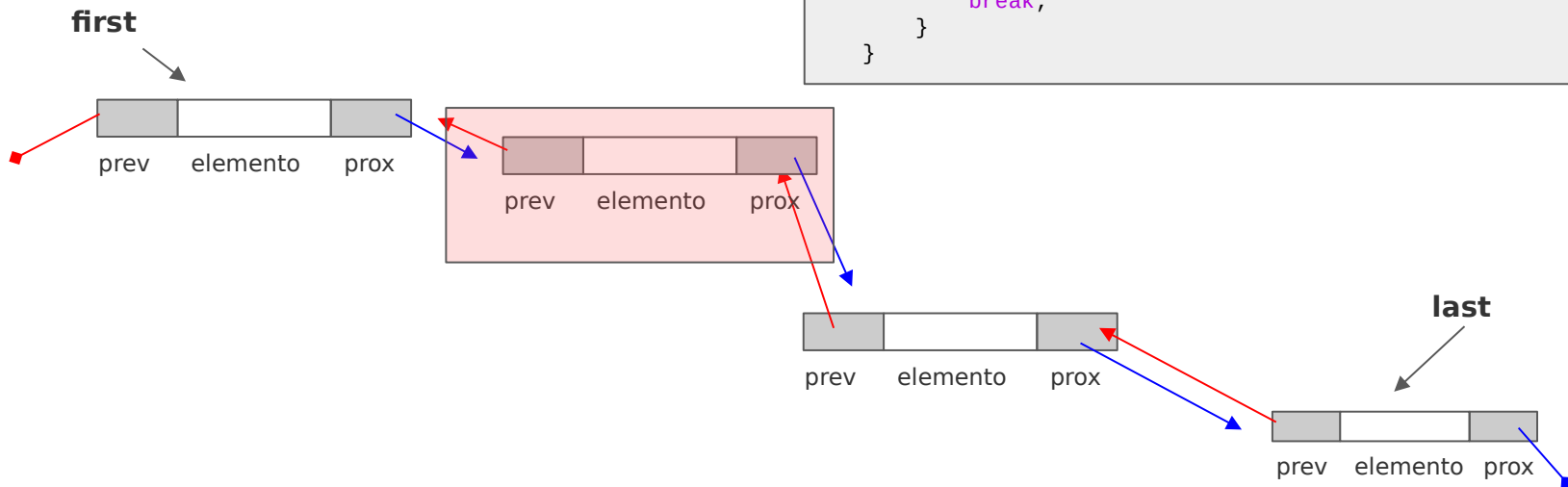
Listas

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



Listas

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



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

Exercícios

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

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

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

- Crie as estruturas indicadas, e crie o first funcionário da lista;
- Adicione um segundo funcionário ao início da lista;
- 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 first 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 first 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 first para o último elemento, e depois do último para o first.
 - 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.