

Listas Duplamente Encadeadas

Prof. Denio Duarte

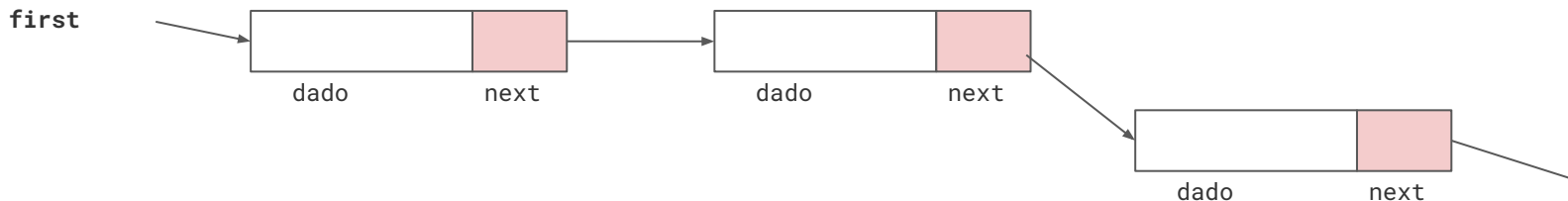
duarte@uffs.edu.br

Prof. Caio V. Koch

caio.santos@uffs.edu.br

Lista encadeada

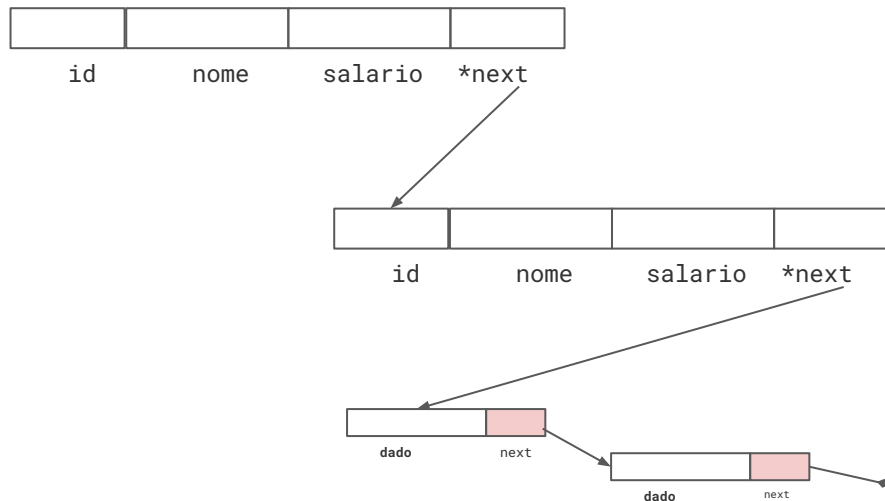
- Uma lista encadeada representa uma sequência de objetos, do mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
 - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista



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

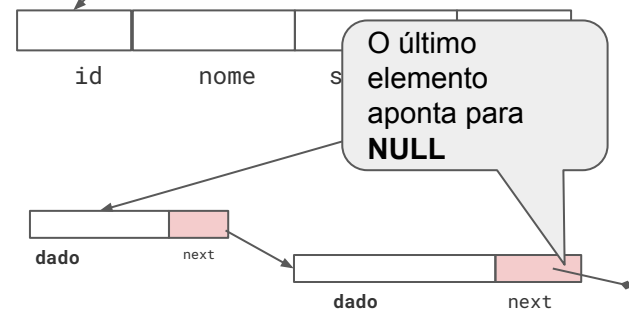
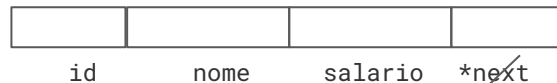
```
struct funcionario{  
    int id;  
    char nome[TAM_NOME+1];  
    double salario;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```



Lista encadeada simples

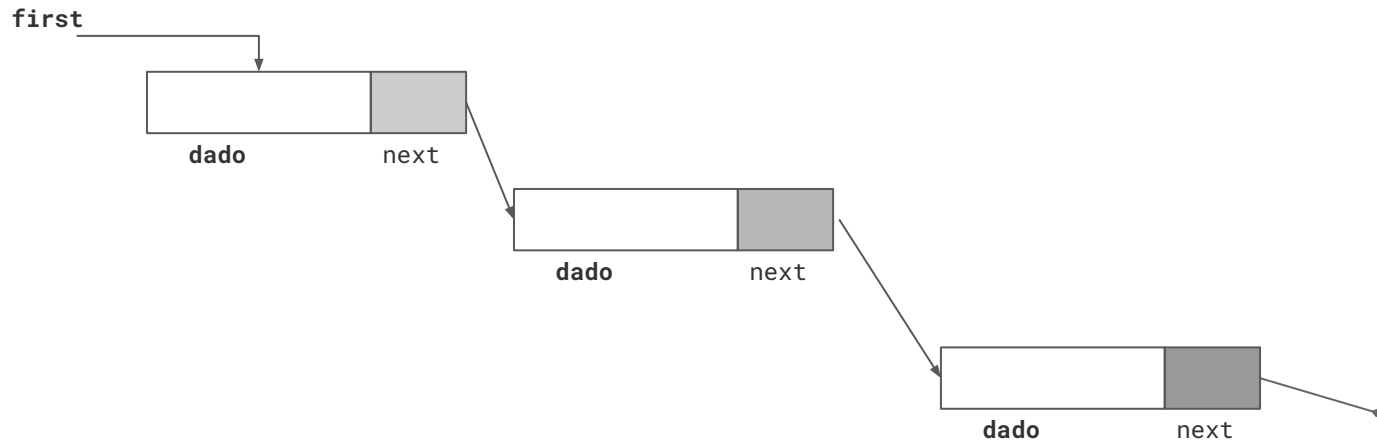
- Uma lista encadeada representa uma sequência de objetos, de mesmo tipo, na memória. Cada elemento da sequência armazena seu valor e o endereço do próximo elemento
 - Ou seja, junto a cada um dos elementos da lista, explicitamente armazenamos o endereço para o próximo elemento da lista
 - Sabe-se quem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



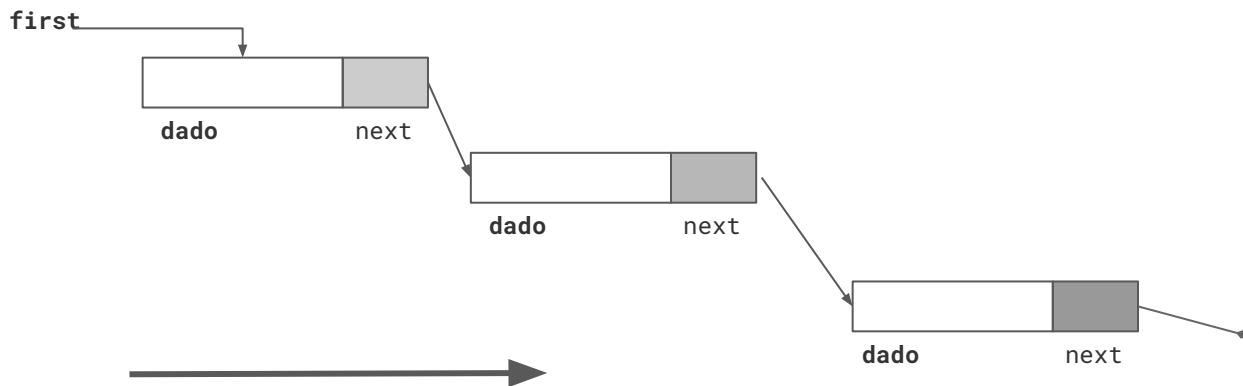
Listas

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



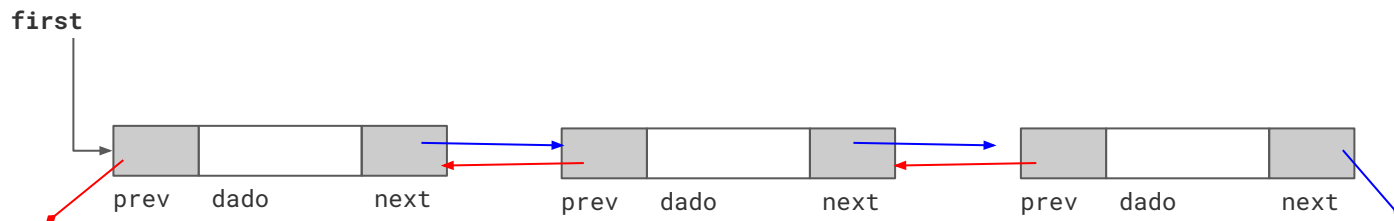
Listas

- 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 um bom desempenho (uso de recursividade)
 - Deletar um elemento da lista também não é trivial, já que precisamos armazenar o elemento anterior.



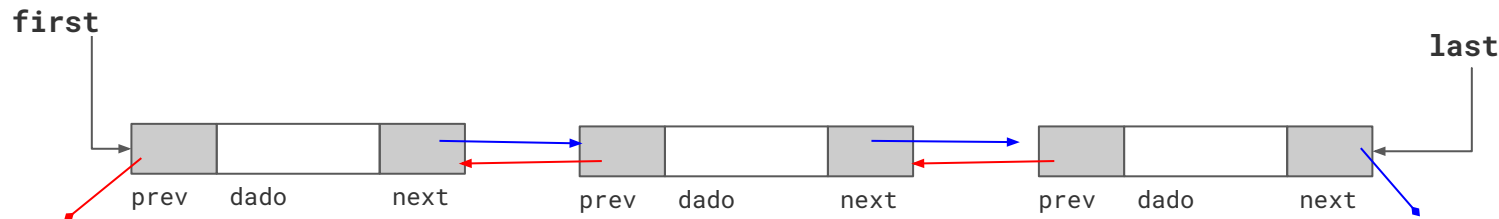
Listas Duplamente Encadeadas

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



Listas Duplamente Encadeadas

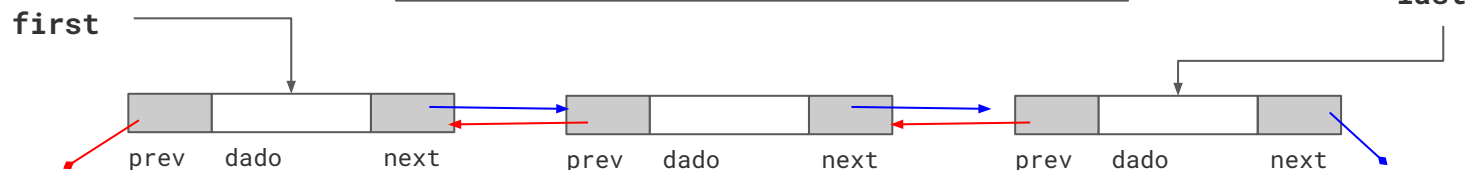
- Em uma lista duplamente encadeada, cada elemento possui um ponteiro para seu anterior e um ponteiro para o seu próximo.
 - Utilizando esta lista, sabemos facilmente o próximo elemento e o elemento anterior
 - Já que temos a estrutura encadeada pelo próximo e pelo anterior podemos **armazenar** o **first** (**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 *prev;  
};  
typedef struct funcionario Funcionario;
```

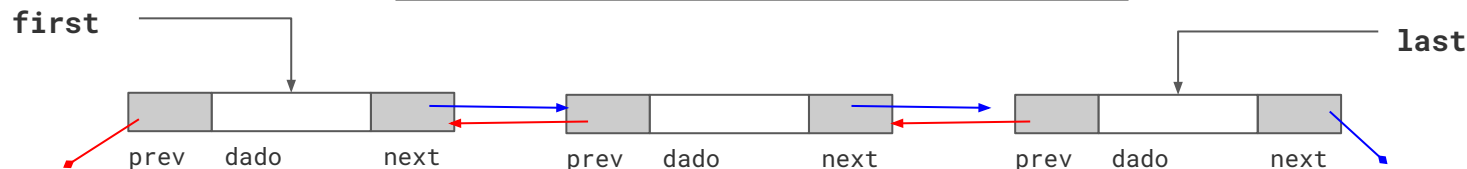


Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

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



Listas Duplamente Encadeadas

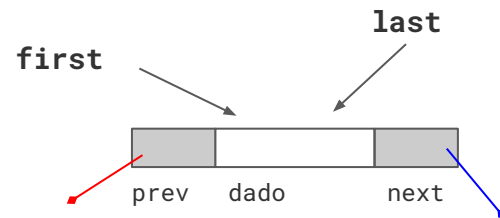
- Inserir um elemento

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

```
Funcionario *first, *last;
first = (Funcionario *)malloc (sizeof(Funcionario));

first->id = 1;
first->idade = 31;
first->salario = 234.0;
first->next = NULL;
first->prev = NULL;

last = first;
```

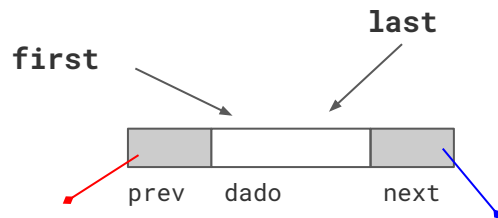


Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

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

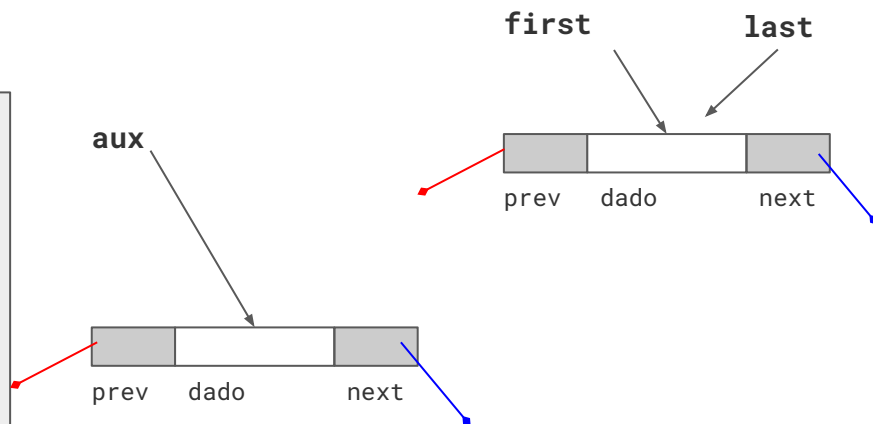


Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

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

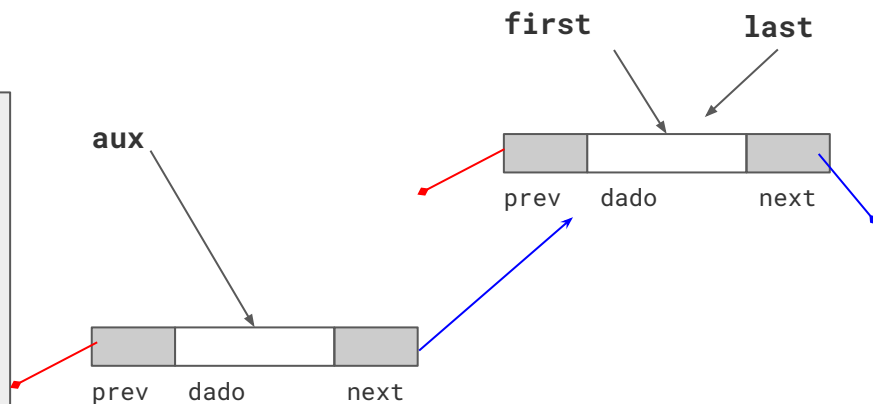


Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

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

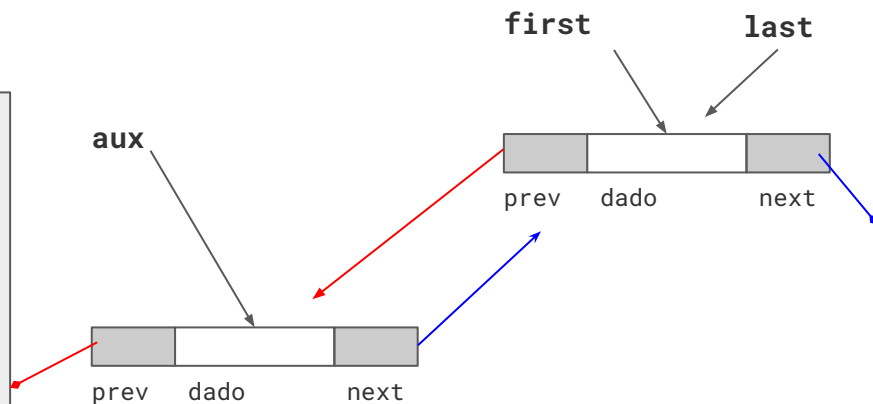


Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

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

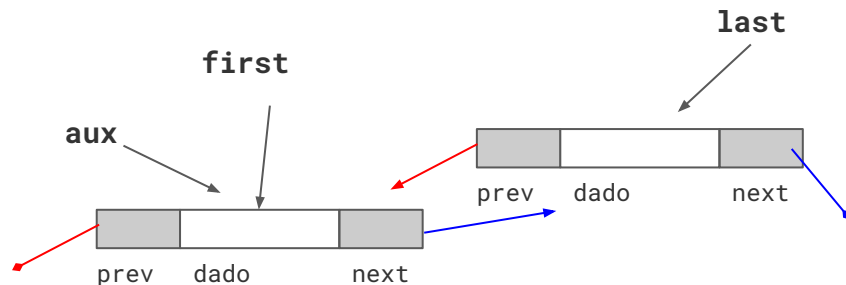


Listas Duplamente Encadeadas

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

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

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



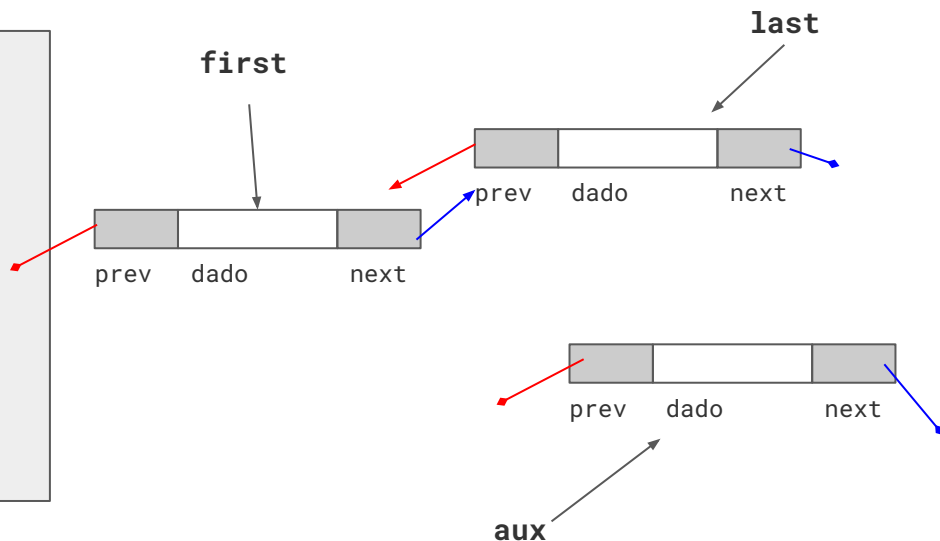
Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux;  
aux = (Funcionario *) malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->prev = NULL;
```

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



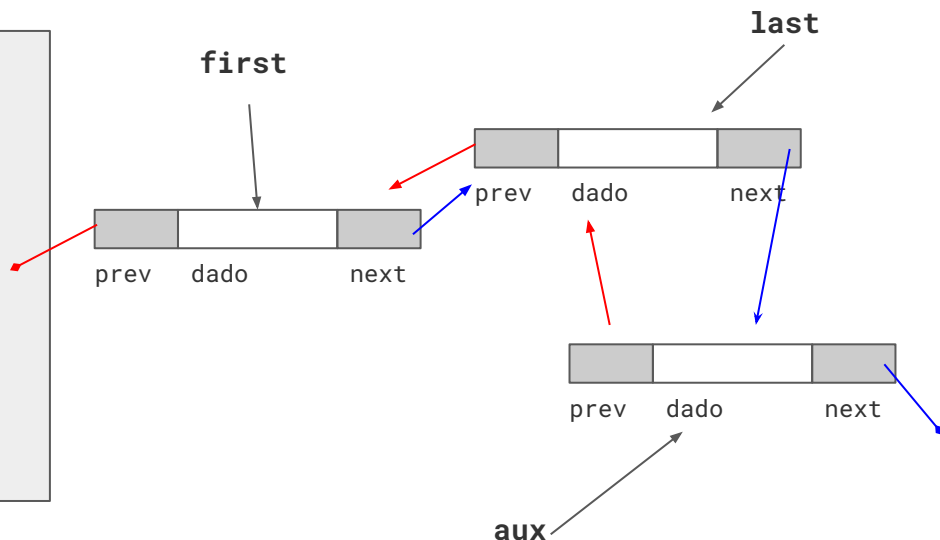
Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux;  
aux = (Funcionario *) malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->prev = NULL;
```

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



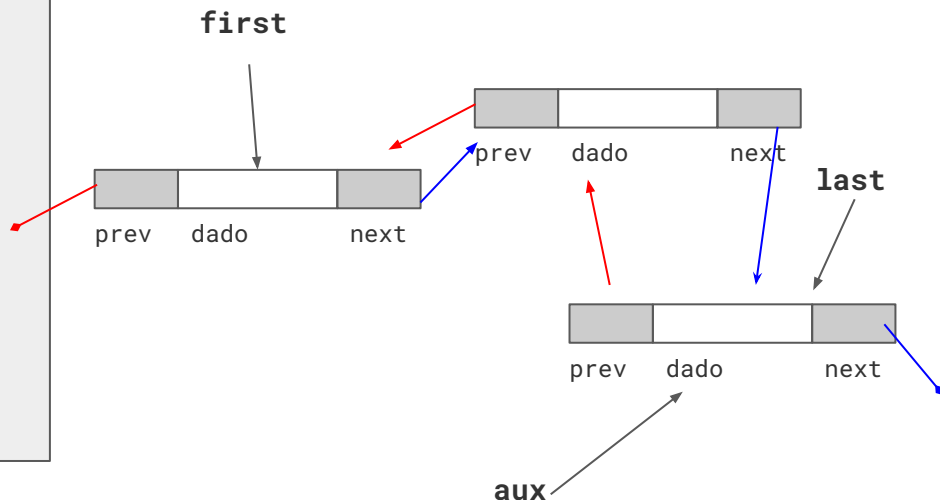
Listas Duplamente Encadeadas

- 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 *prev;  
};  
typedef struct funcionario Funcionario;
```

```
Funcionario *aux;  
aux = (Funcionario *) malloc (sizeof(Funcionario));  
aux->id = i+1;  
aux->idade = 39;  
aux->salario = 234.0;  
aux->next = NULL;  
aux->prev = NULL;
```

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

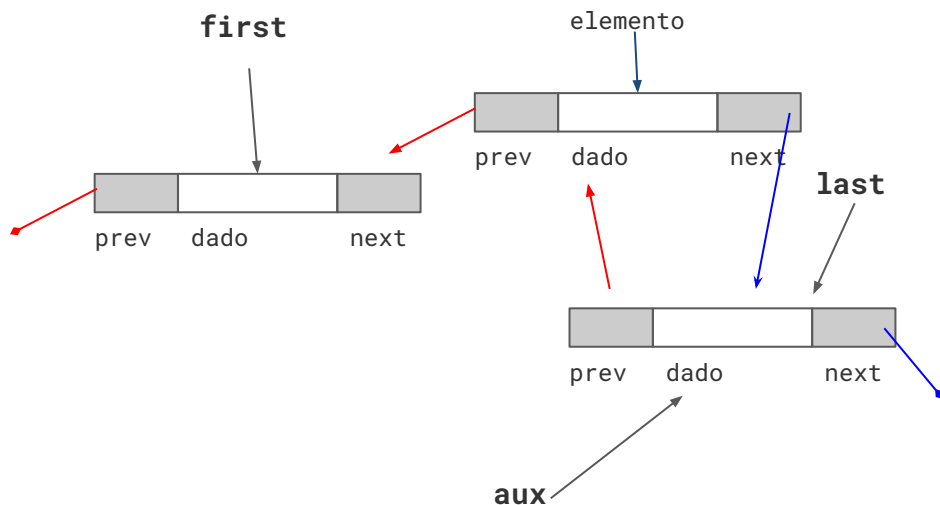


Listas Duplamente Encadeadas

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

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

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

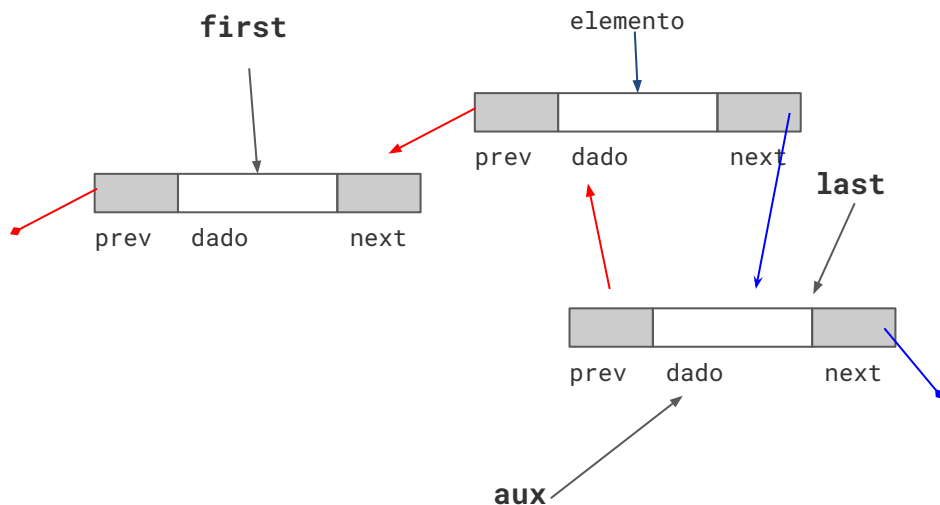


Listas Duplamente Encadeadas

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

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



Listas Duplamente Encadeadas

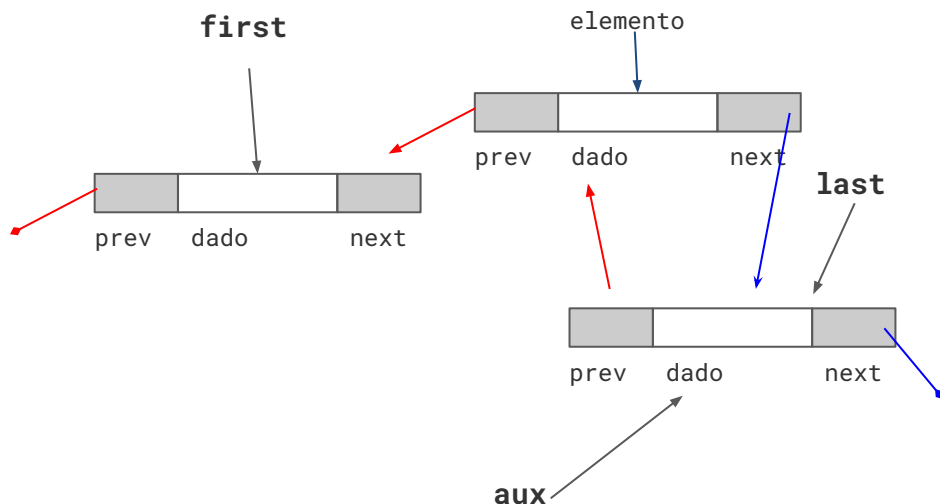
- Se eu adicionar mais um, onde adiciono

- Início da lista
- Fim da lista
- No meio

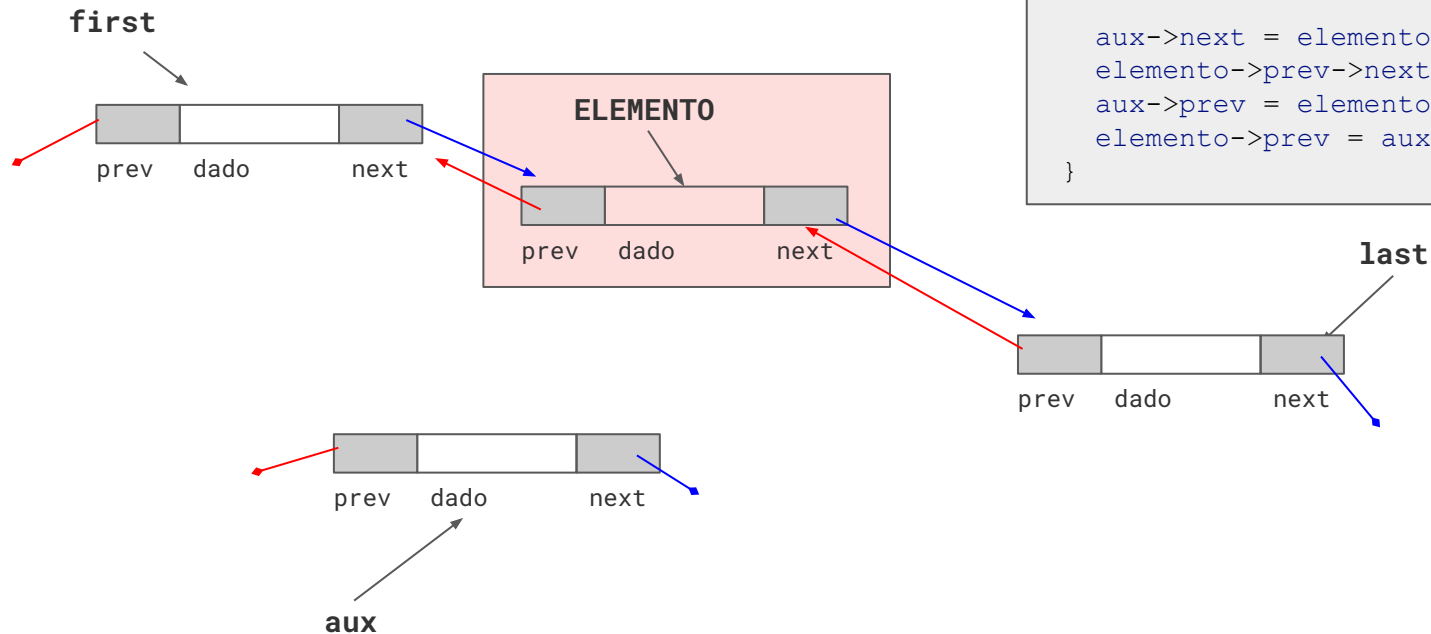
magia() não é uma função real, estamos usando para ilustrar que sabemos qual a posição correta.

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

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



Listas Duplamente Encadeadas

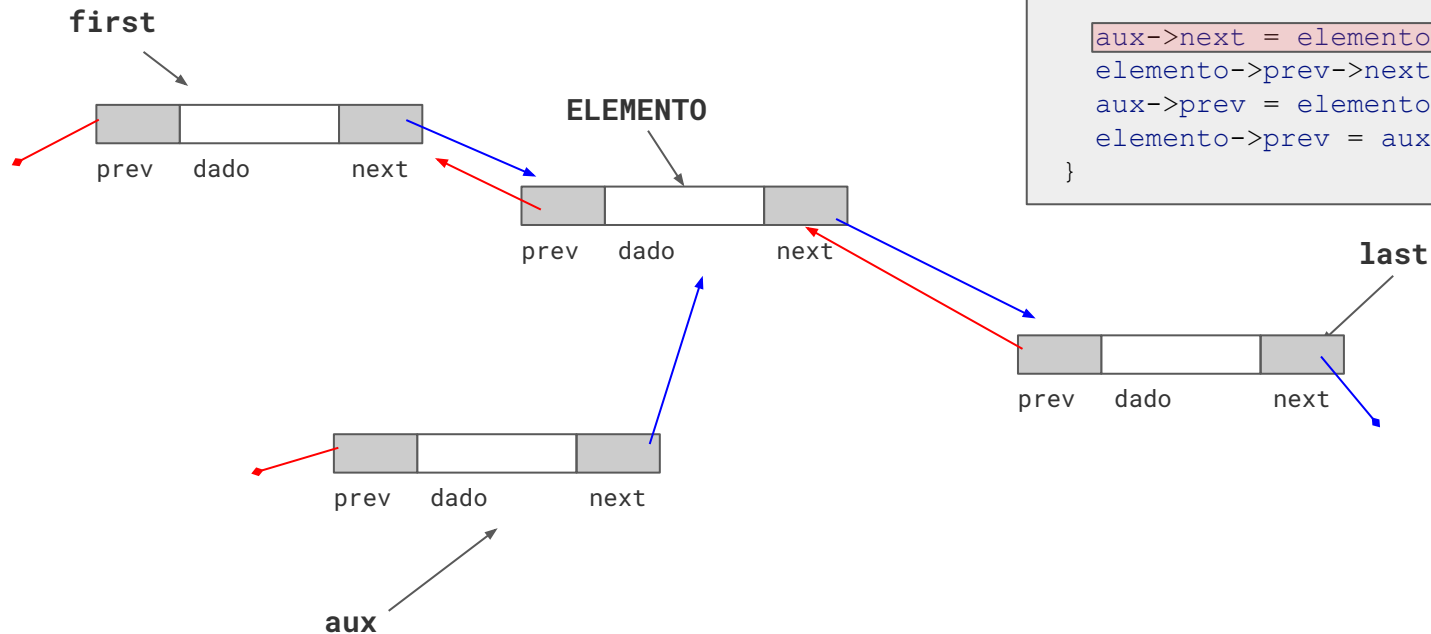


```
Funcionario *aux;
for (i = 1; i < 10; i++) {

    //mágica que encontra a posição do elemento
    Funcionario *elemento = magia();
    aux = malloc (sizeof(Funcionario));
    aux->id = i;
    aux->idade = 39;
    aux->salario = 234.0;
    aux->next = NULL;
    aux->prev = NULL;

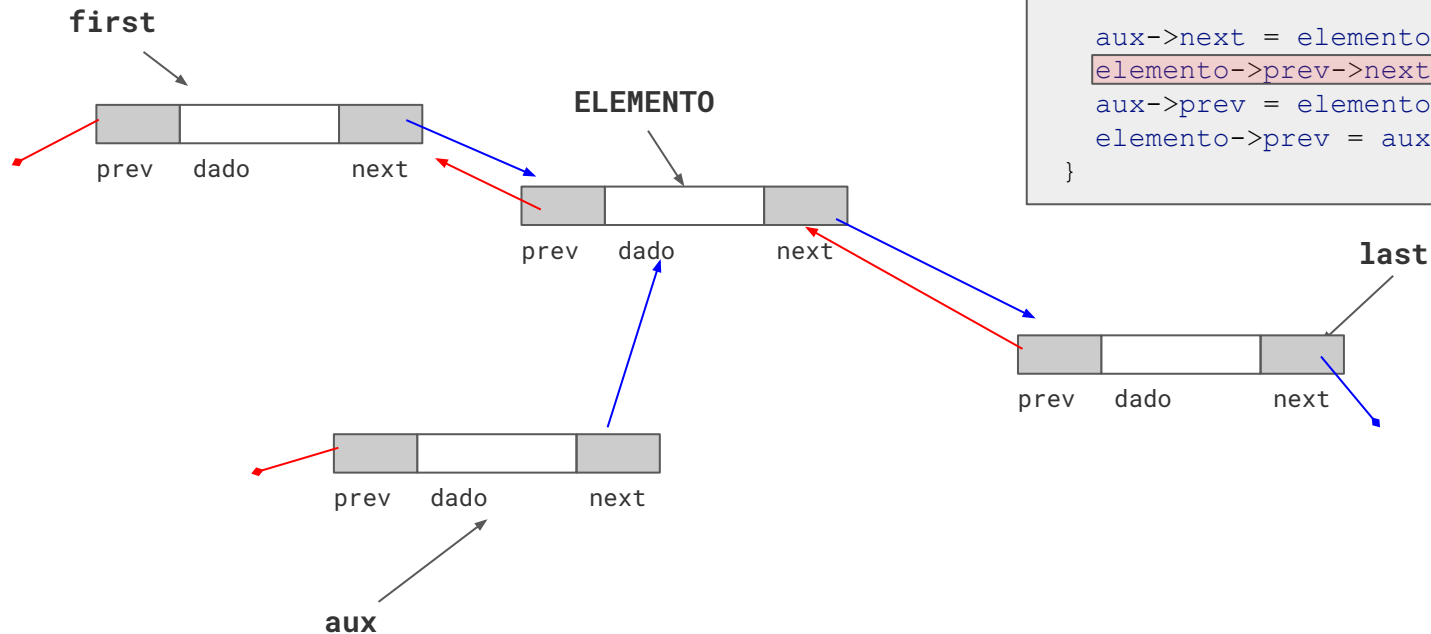
    aux->next = elemento;
    elemento->prev->next = aux;
    aux->prev = elemento->prev;
    elemento->prev = aux;
}
```

Listas



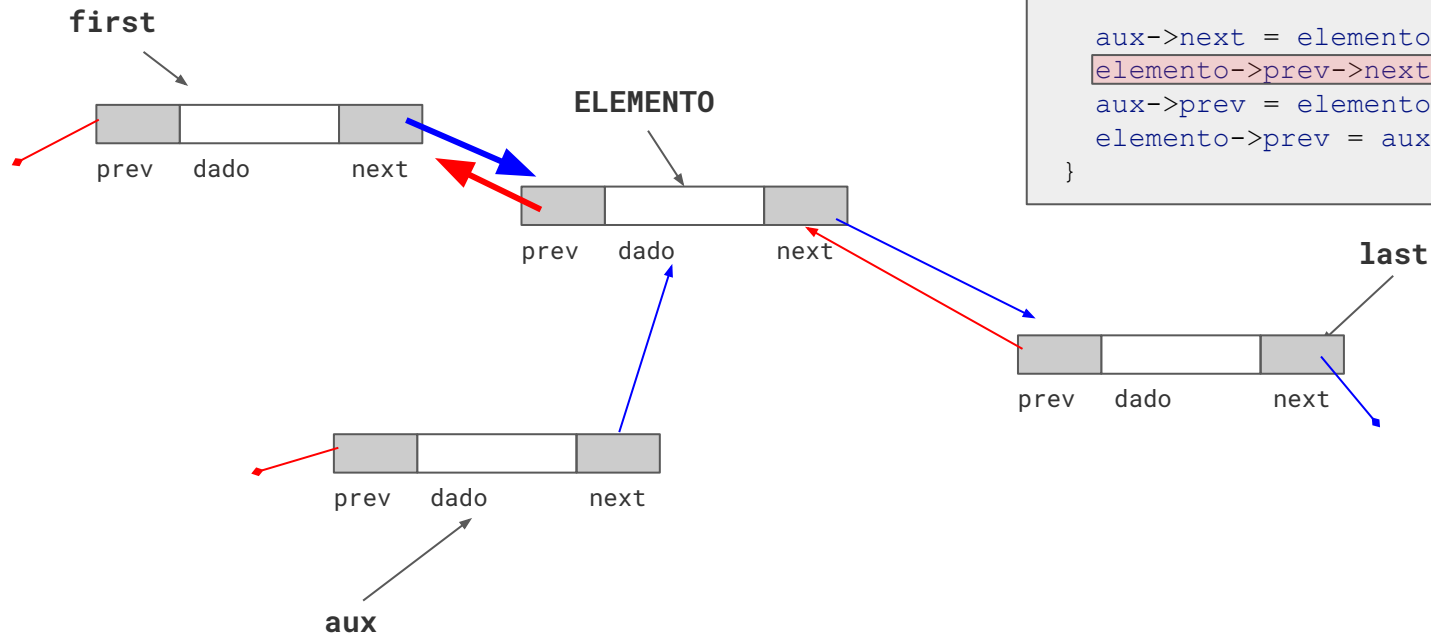
```
Funcionario *aux;  
for (i = 1; i < 10; i++) {  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->prev = NULL; 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

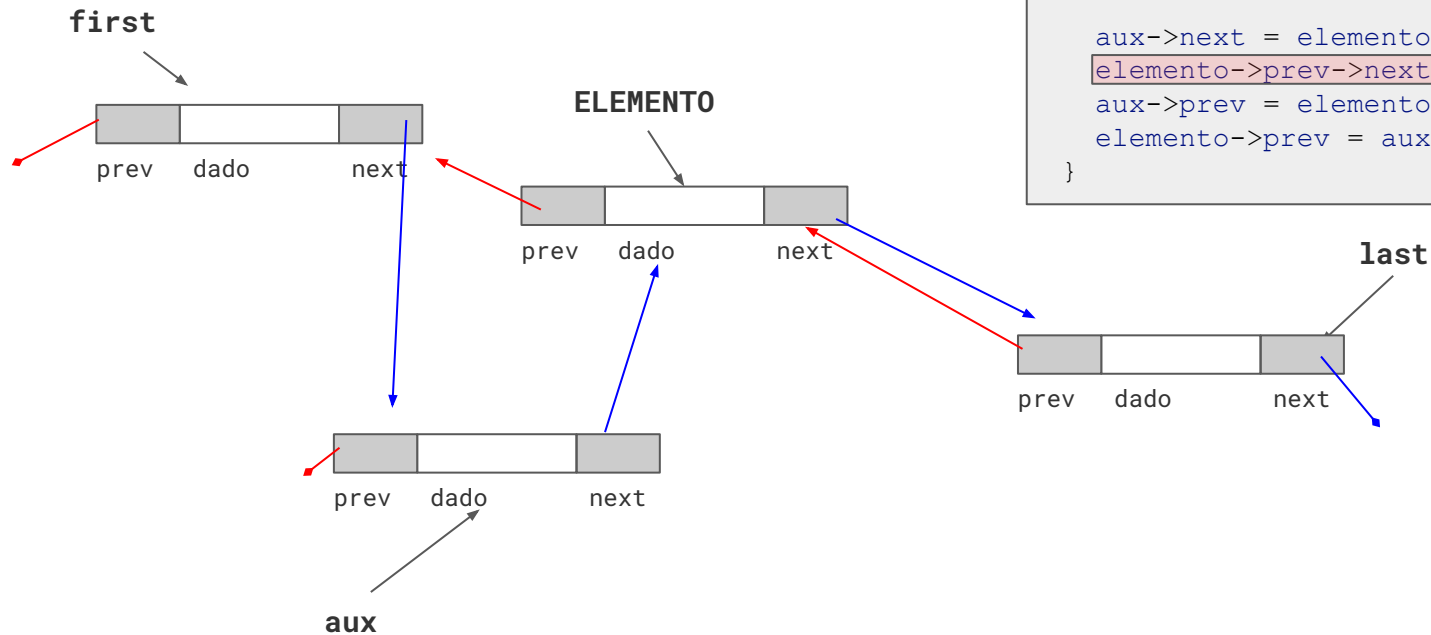
Listas



```
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;
    aux->idade = 39;
    aux->salario = 234.0;
    aux->next = NULL;
    aux->prev = NULL;

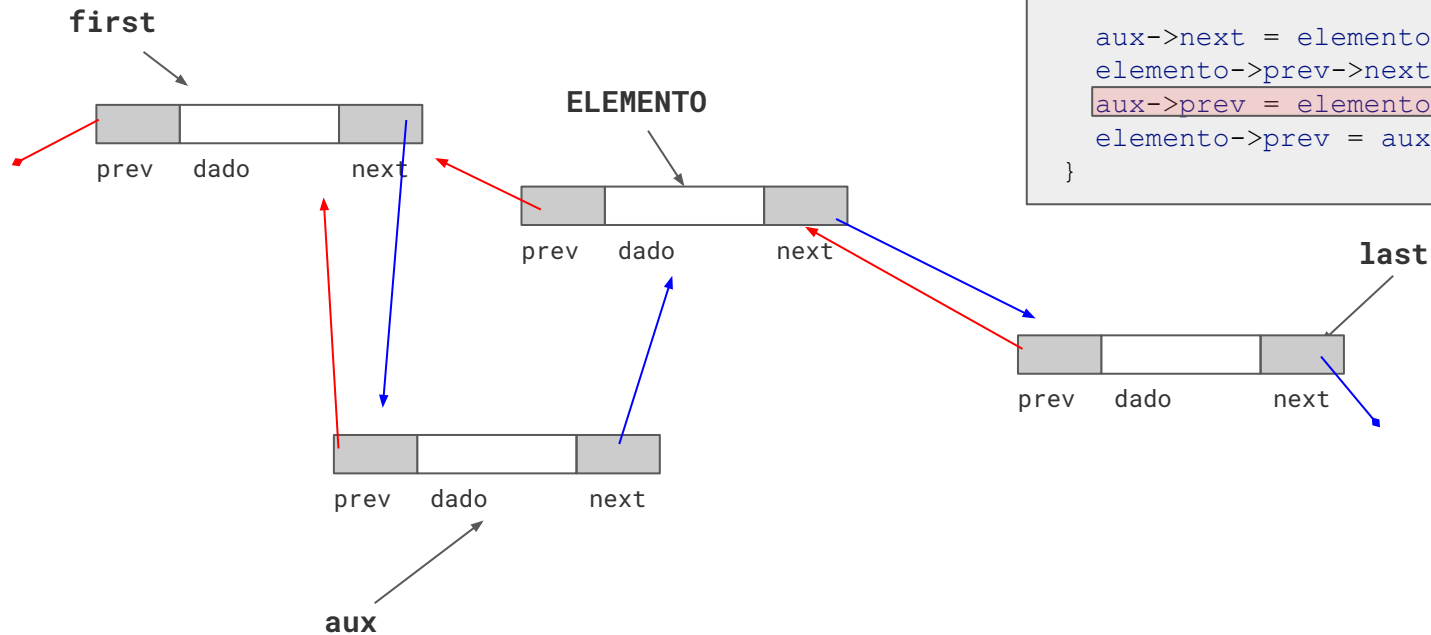
    aux->next = elemento;
    elemento->prev->next = aux;
    aux->prev = elemento->prev;
    elemento->prev = aux;
}
```

Listas



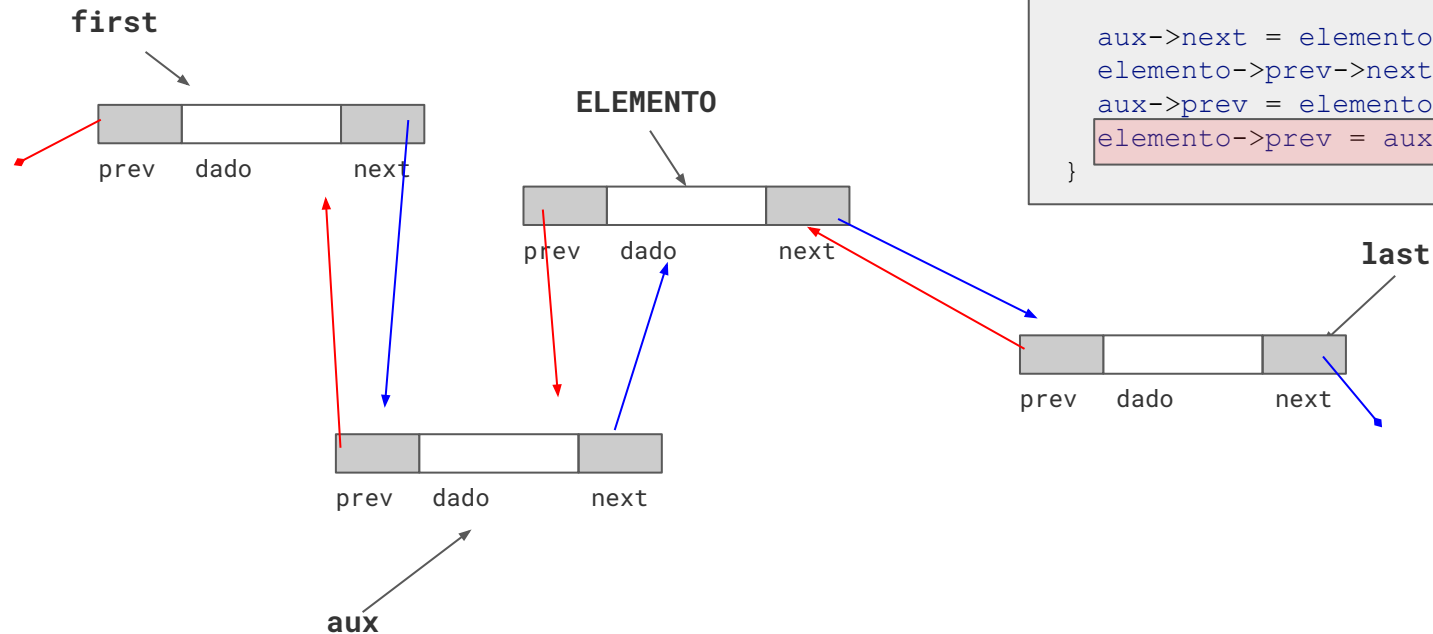
```
Funcionario *aux;  
for (i = 1; i < 10; i++) {  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->prev = NULL; 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

Listas



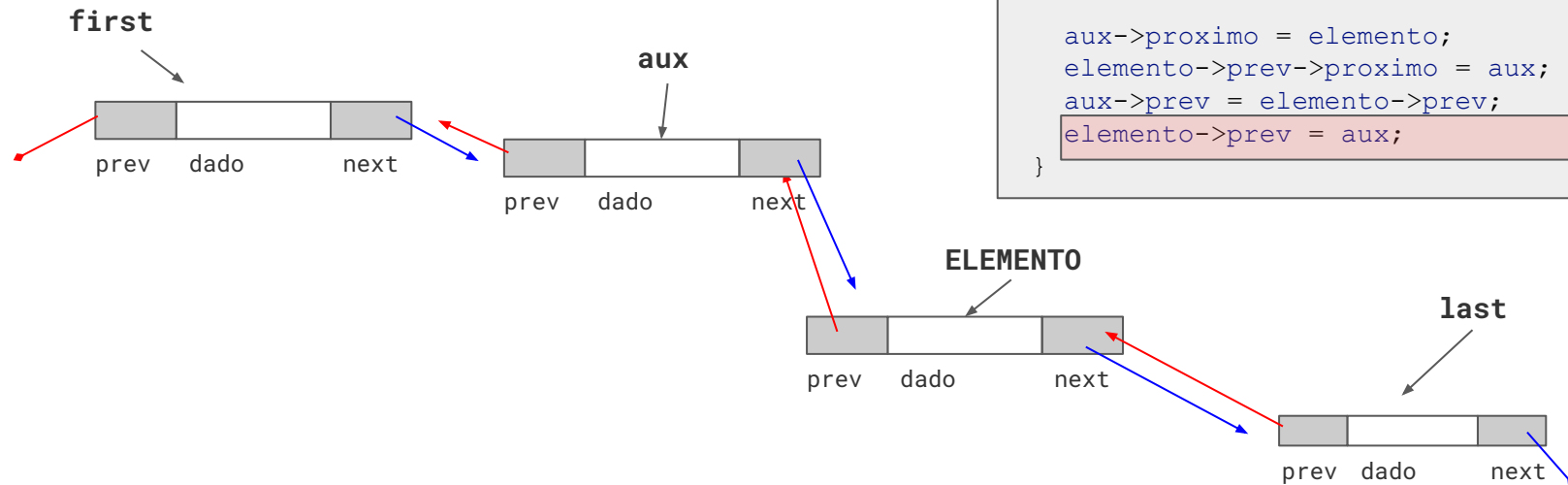
```
Funcionario *aux;  
for (i = 1; i < 10; i++){  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->prev = NULL;  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

Listas



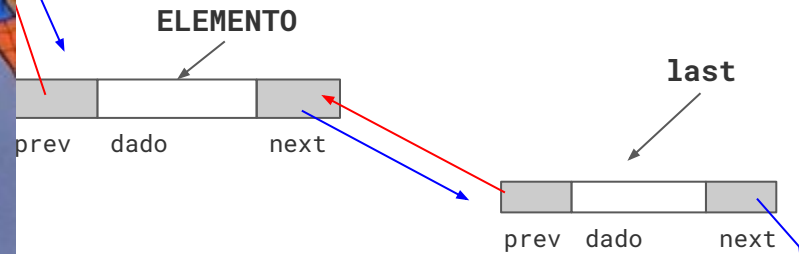
```
Funcionario *aux;
for (i = 1; i < 10; i++) {
    //mágica que encontra a posição do elemento
    Funcionario *elemento = magia();
    aux = malloc (sizeof(Funcionario));
    aux->id = i;
    aux->idade = 39;
    aux->salario = 234.0;
    aux->next = NULL;
    aux->prev = NULL;

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

Listas

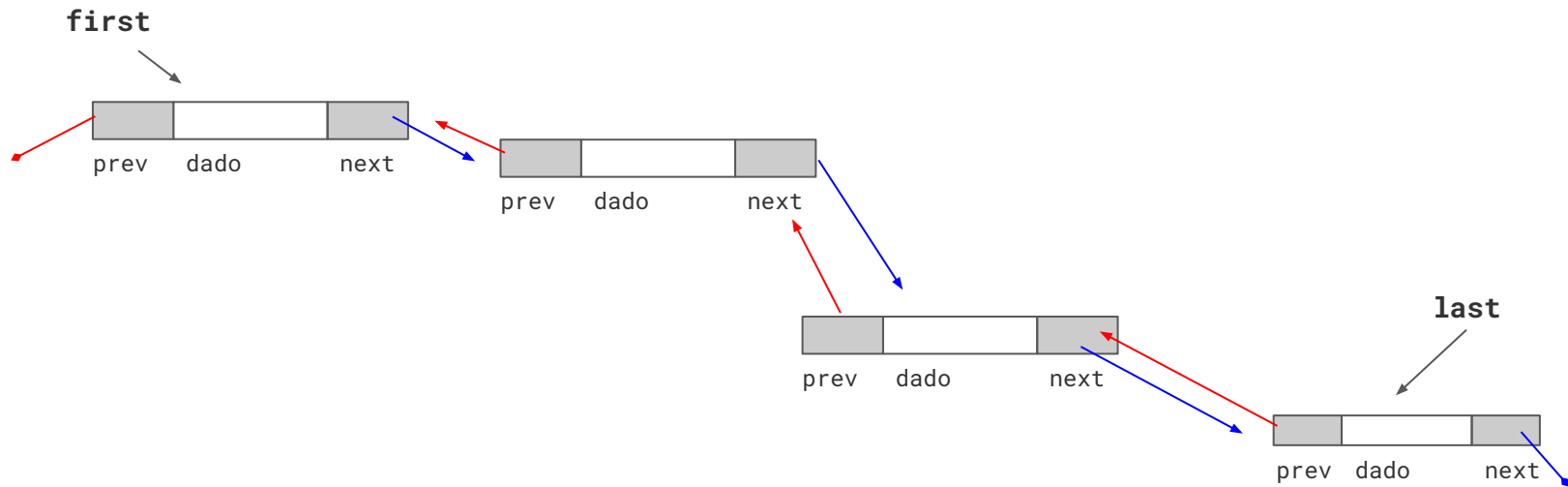


```
Funcionario *aux;  
for (i = 1; i < 10; i++){  
    //mágica que encontra a posição do elemento  
    Funcionario *elemento = magia();  
    aux = malloc (sizeof(Funcionario));  
    aux->id = i;  
    aux->idade = 39;  
    aux->salario = 234.0;  
    aux->next = NULL;  
    aux->prev = NULL;  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



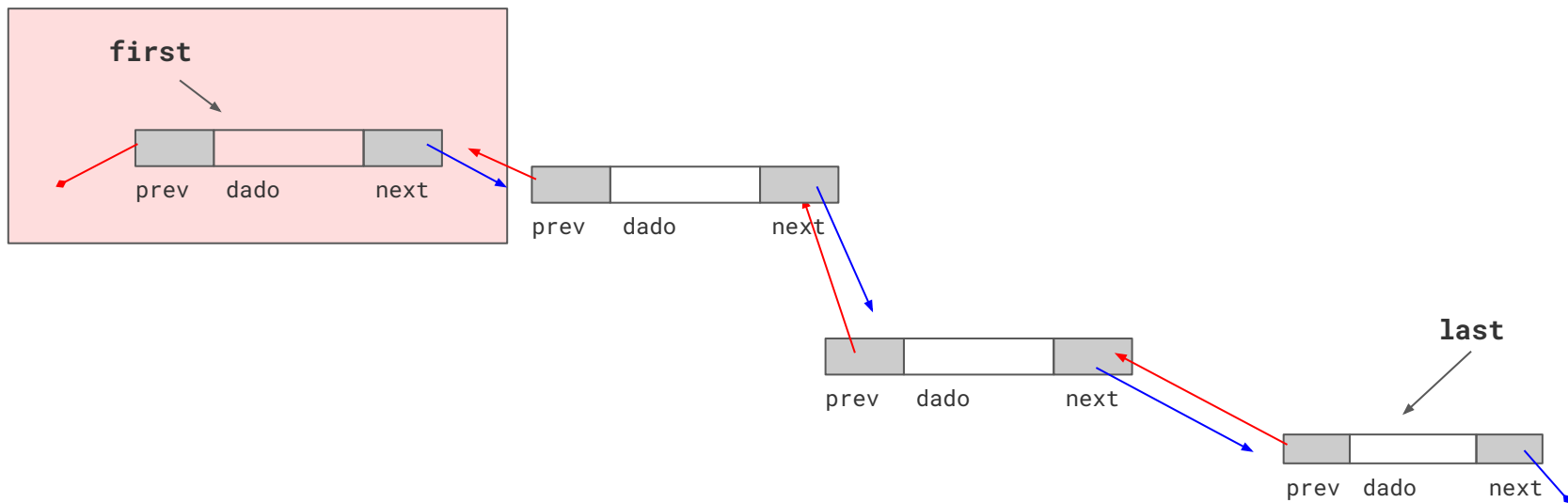
Listas

- Deletar um item



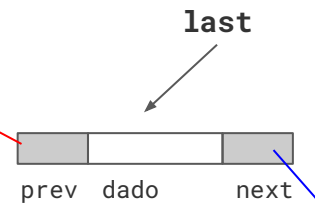
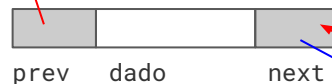
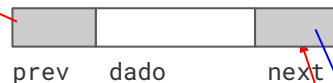
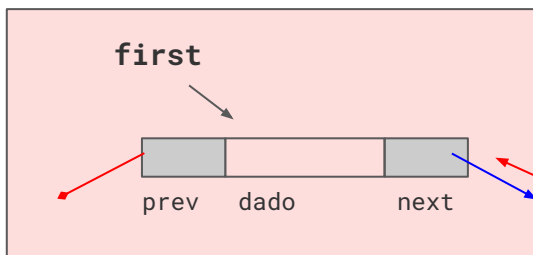
Listas

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



Listas

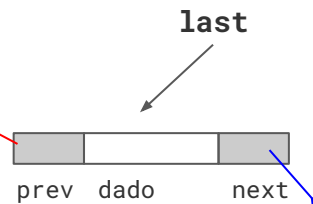
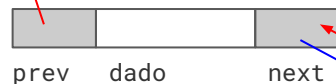
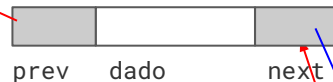
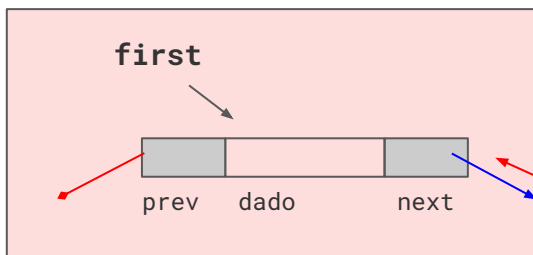
- Para eliminar um item devemos considerar:
 - Eliminar do início



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

Listas

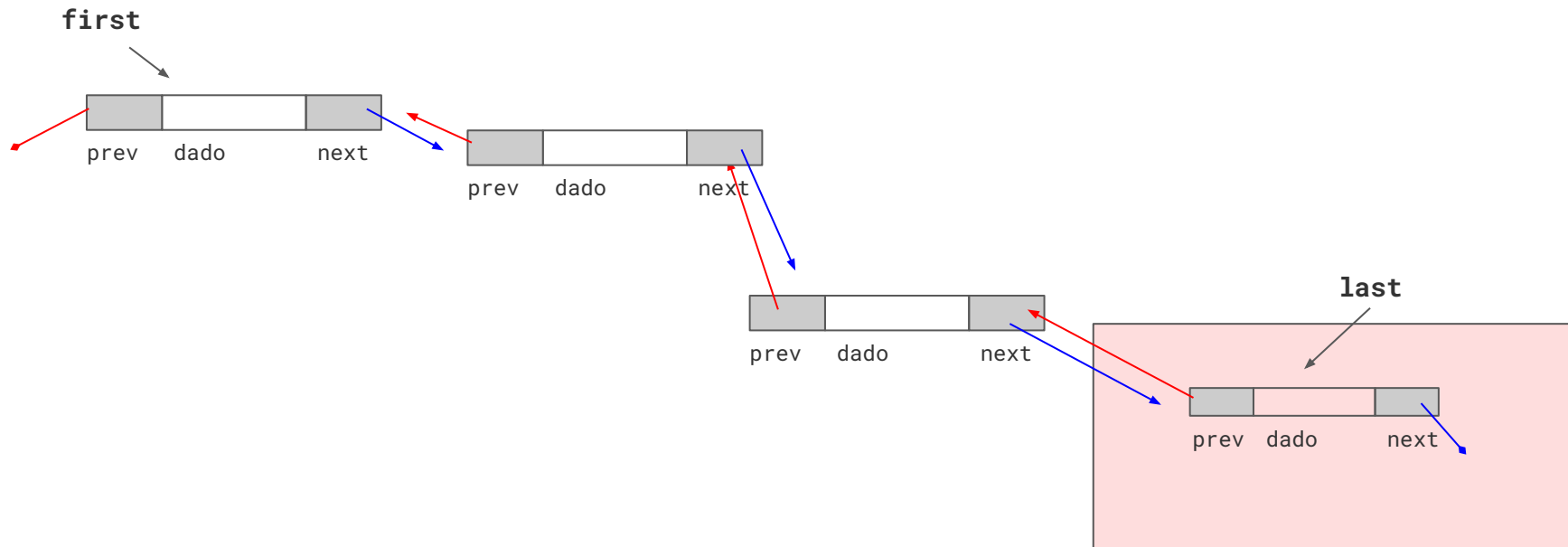
- Para eliminar um item devemos considerar:
 - Eliminar do início



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

Listas

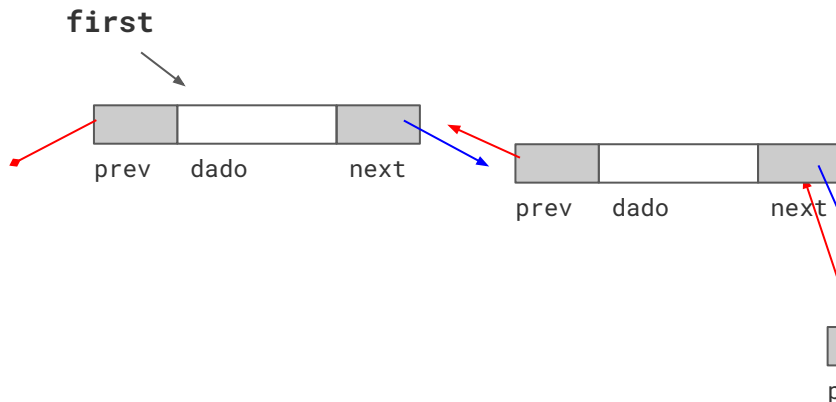
- Para eliminar um item devemos considerar 3 casos
 - Eliminar do início
 - Eliminar do fim



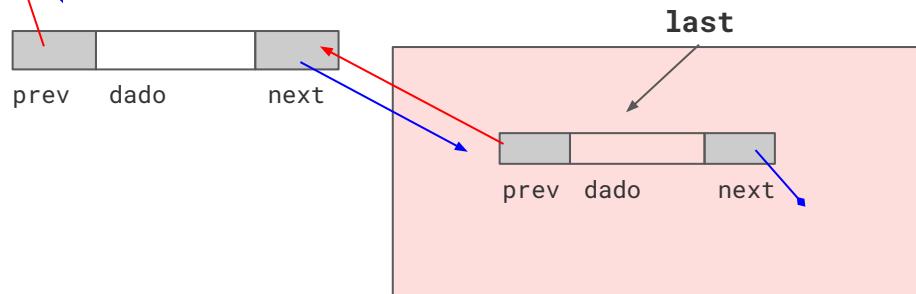
Listas

- Para eliminar um item devemos considerar:

- Eliminar do início
- Eliminar do fim

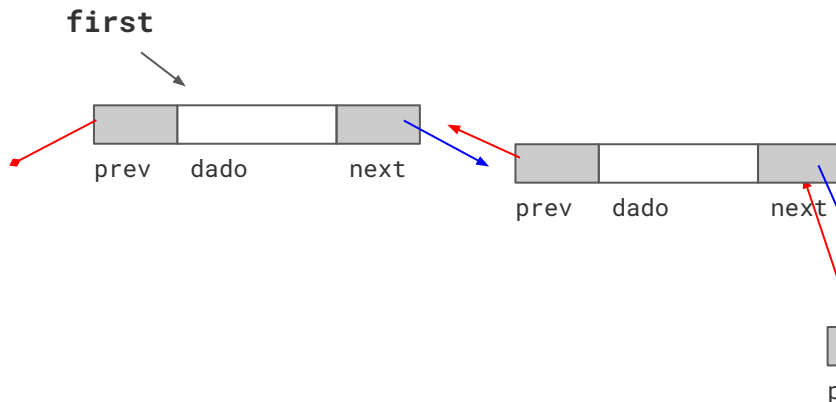


```
Funcionario *aux, *previous; //vai ser nosso 'contador'
int idDelete; //id do funcionario a ser apagado
for (aux = first; aux != NULL; aux = aux->prox){
    if (aux->id == idDelete){
        if (aux == first) { //verifica se é o first
            first = first->next; /
            first->prev = NULL;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

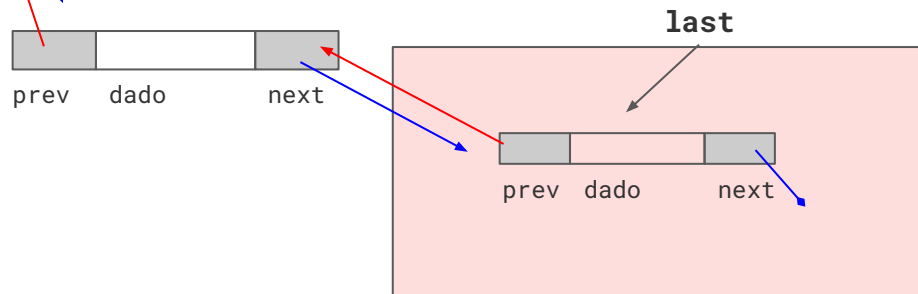


Listas

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

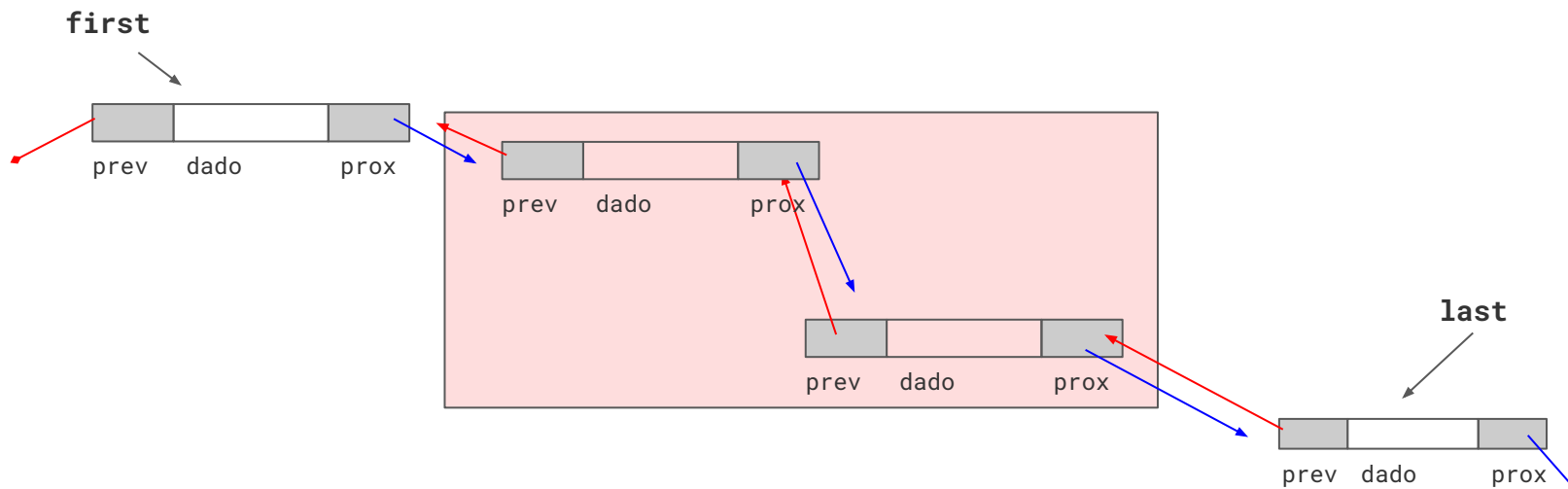


```
Funcionario *aux;  
int idDelete; //id do funcionario a ser apagado  
for (aux = first; aux != NULL; aux = aux->prox) {  
    if (aux->id == idDelete) {  
        if (aux == first) { //verifica se é o first  
            first = first->next; /  
            first->prev = NULL;  
        } if (aux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



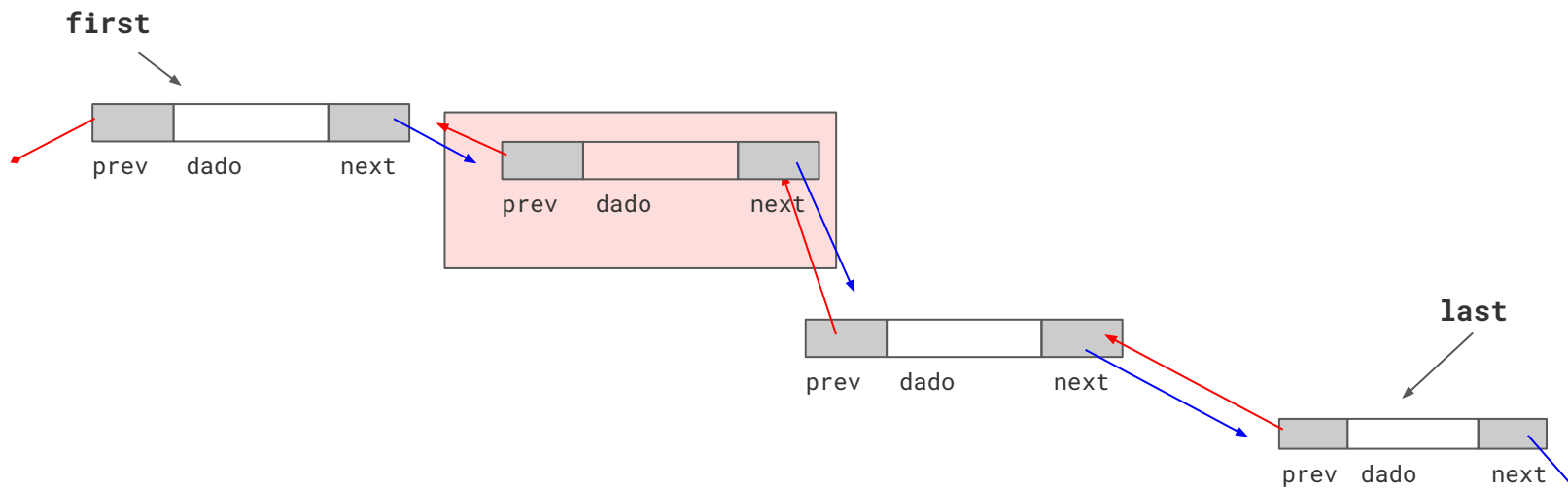
Listas

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



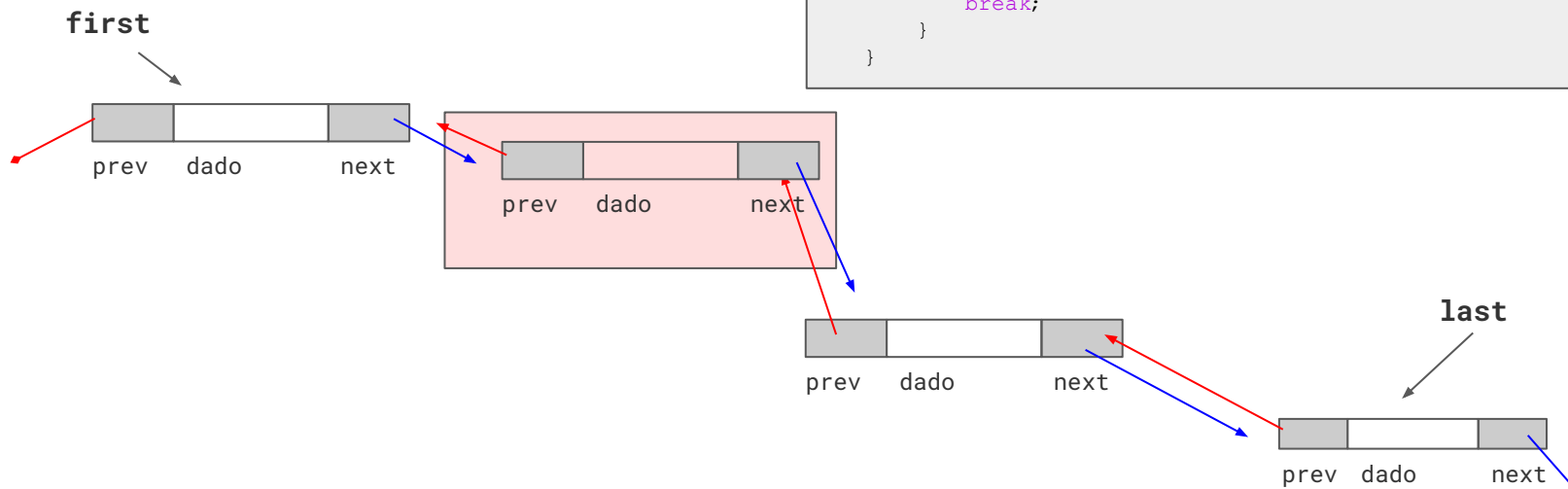
Listas

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



Listas

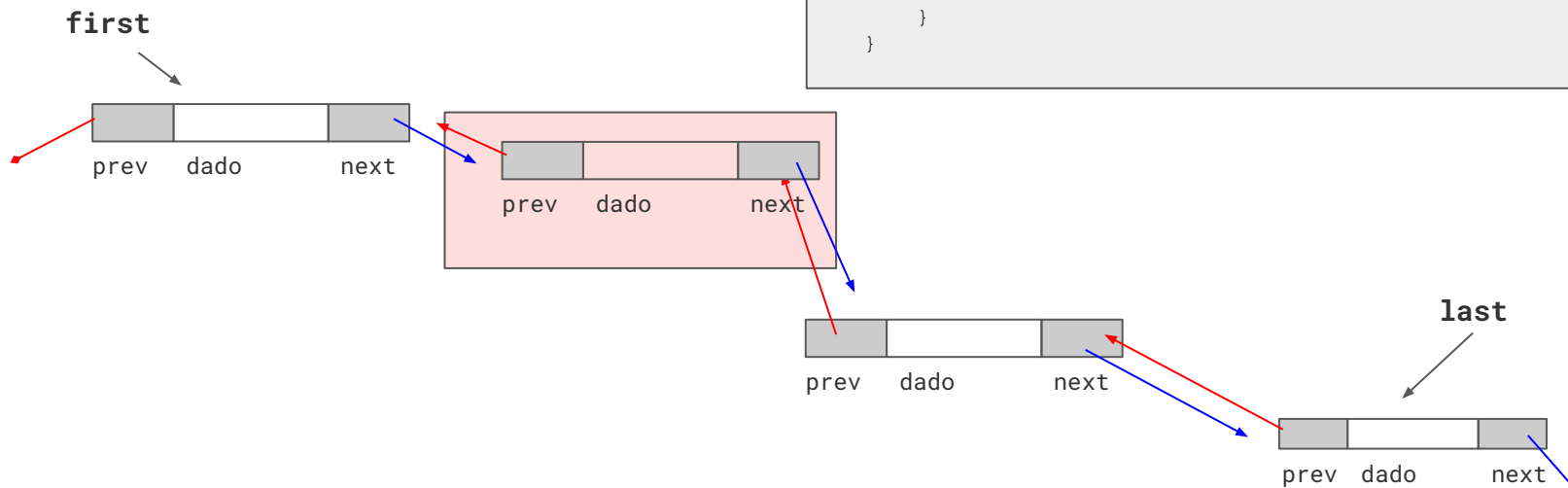
- Para eliminar um item devemos considerar:
 - Eliminar do início
 - Eliminar do fim
 - Eliminar do meio



```
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->prev = NULL;
        } if (aux == last) { //verifica se é o last
            last = last->prev;
            last->next = NULL;
        } else {
            previous = aux->prev;
            previous->next = aux->next;
            previous->next->prev = previous;
        }
        free(aux); //apaga o aux
        break;
    }
}
```

Listas

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



```
Funcionario *aux;  
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->prev = NULL;  
        } if (aux == last) { //verifica se é o last  
            last = last->prev;  
            last->next = NULL;  
        } else {  
            aux->prev->next = aux->next;  
            aux->next->prev = aux->prev;  
        }  
        free(aux); //apaga o aux  
        break;  
    }  
}
```

Outra forma de
percorrer e excluir um
nó da lista.

Excluir o primeiro nó

Excluir o último nó

Excluir nó intermediário

```
1  Funcionario *posicaoExcluir = first;
2  int idDelete = 1;
3  //Percorre a lista até encontrar o nó a excluir, ou NULL
4  while(posicaoExcluir != NULL){
5      if(posicaoExcluir->id == idDelete) break;
6      posicaoExcluir = posicaoExcluir->next;
7  }
8
9  if(posicaoExcluir != NULL){ //então tem um nodo a excluir
10     //verificar se é o primeiro nó
11     if(posicaoExcluir->prev == NULL){ //então é o primeiro nó da lista
12         first = first->next; //atualiza a cabeça da lista
13         if(first != NULL){ //há mais nós na lista (à direita)
14             first->prev = NULL; //mas não há nós à esquerda
15         } else {last = NULL;} //se houver uma variável last
16     } else if(posicaoExcluir->next == NULL){ //excluir último nó da l
17         posicaoExcluir->prev->next = NULL;
18         //atualizar o last, se houver
19         last = posicaoExcluir->prev;
20     } else { //então é um nó intermediário
21         posicaoExcluir->prev->next = posicaoExcluir->next;
22         posicaoExcluir->next->prev = posicaoExcluir->prev;
23     }
24     free(posicaoExcluir);
25 } else {
26     printf("Id não encontrado!\n");
27 }
```

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 *prev;  
    struct funcionario *next;  
};  
typedef struct funcionario Funcionario;
```

- Crie uma lista e o primeiro funcionário da lista, considerando as estruturas indicadas;
- Adicione um segundo funcionário no 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 no final 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 duplamente 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 em que 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. Crie uma função de busca que apresenta (imprime na tela) as informações de um funcionário. A busca deve ser feita utilizando o id.