ESTRUTURA DE DADOS

Curso de Licenciatura em Ciências da Computação

Lucas Sampaio Leite

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Listas implementadas a partir de vetores são interessantes porque oferecem acesso rápido aos elementos, já que utilizam blocos contíguos de memória, permitindo acesso direto por índice em tempo constante (O(1));
- No entanto, possuem tamanho fixo, o que implica algumas limitações:
  - A quantidade de elementos não pode ultrapassar o tamanho previamente declarado;
  - Pode haver desperdício de memória caso o vetor seja maior do que o necessário;
  - O redimensionamento é custoso, pois exige a criação de um novo vetor e a cópia dos elementos existentes.

Listas encadeadas

INSTITUTO FEDERAL
Baiano

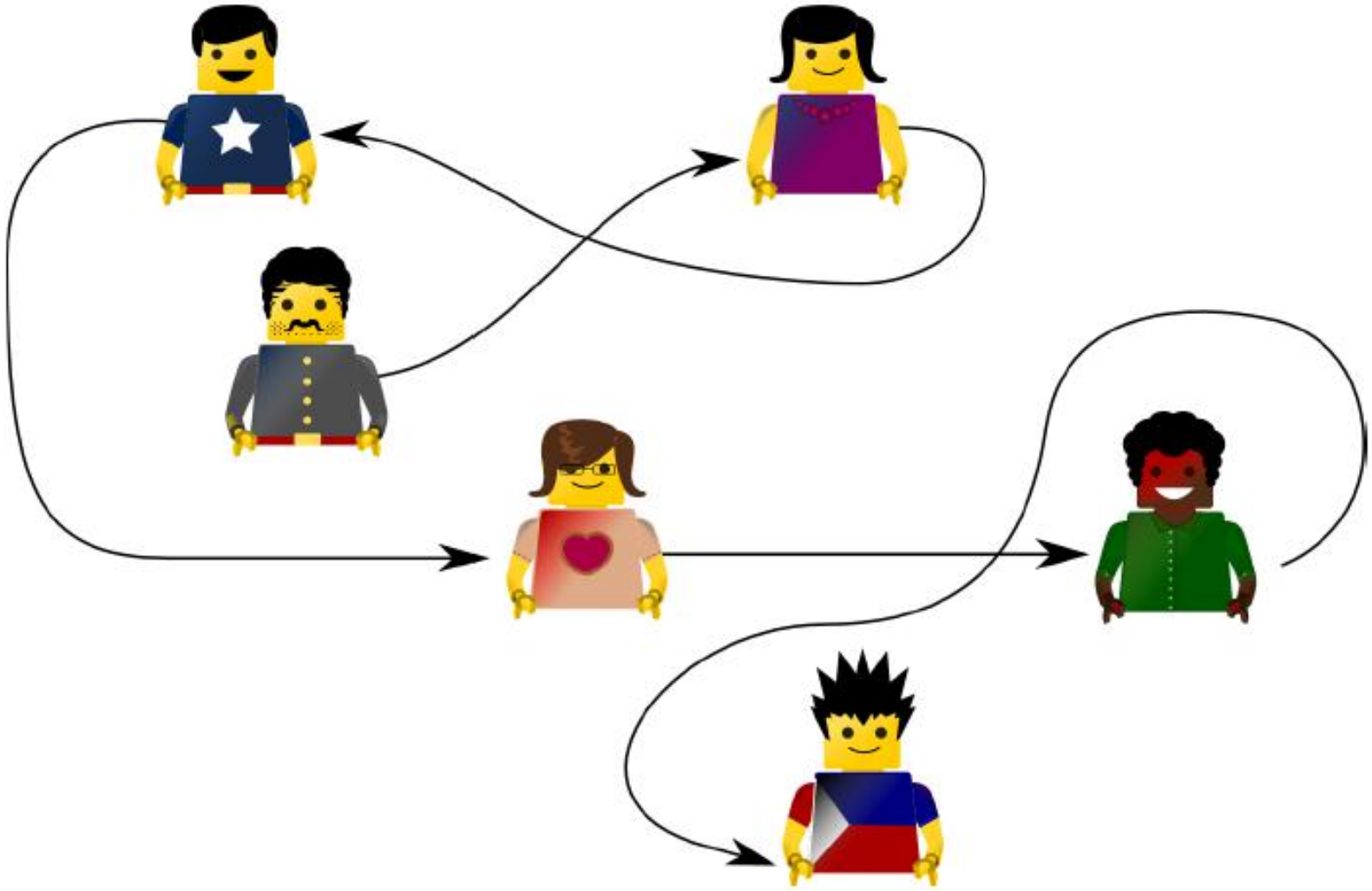
- Adicionar ou remover um elemento na primeira posição exige o deslocamento dos demais elementos (shift right ou shift left), pois é necessário reorganizar as posições ocupadas no vetor:
  - A performance dessas operações se degrada à medida que a quantidade de elementos cresce, já que o deslocamento pode envolver vários elementos (complexidade O(n)).
- Dependendo do problema a ser solucionado, pode ser necessária uma implementação de lista em que a operação de inserir ou remover elementos no início seja computacionalmente eficiente:
  - Nesse caso, torna-se necessário utilizar uma estrutura dinâmica, cujo tamanho não seja fixo e que não dependa de blocos contíguos de memória.

Listas encadeadas

INSTITUTO FEDERAL
Baiano

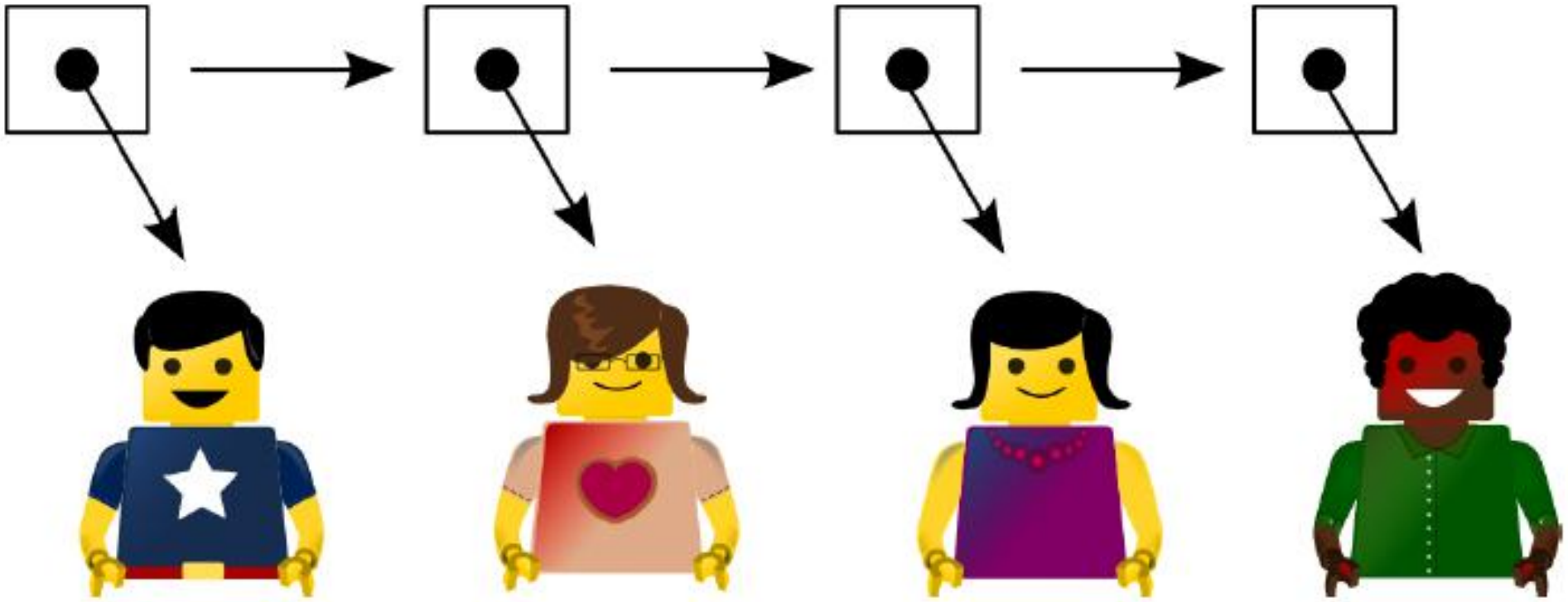
Listas encadeadas

INSTITUTO FEDERAL
Baiano



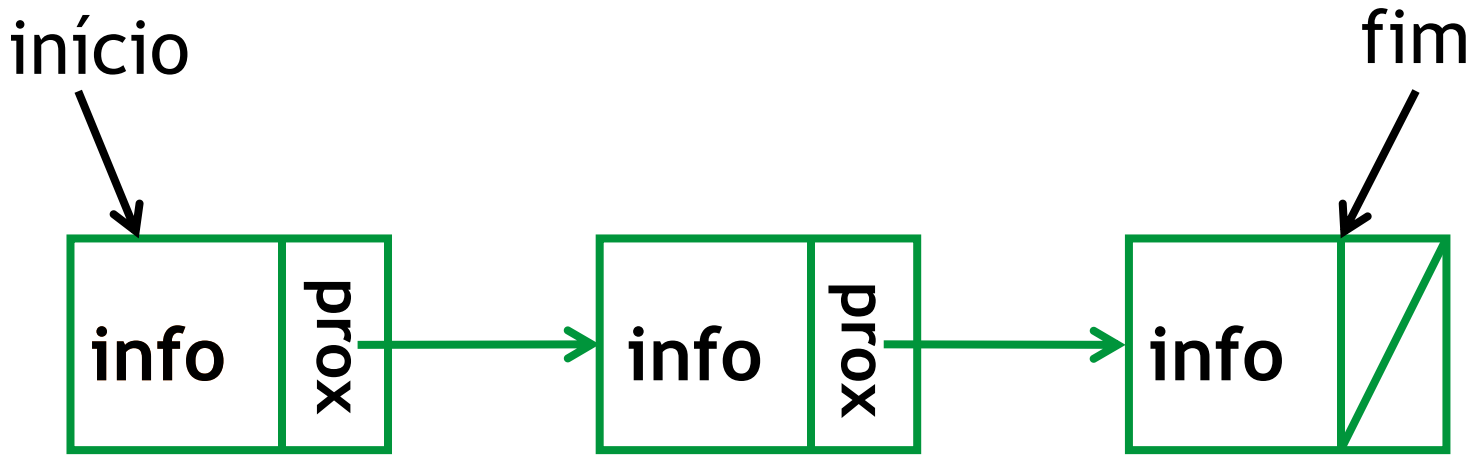
Listas encadeadas

INSTITUTO FEDERAL
Baiano



Listas encadeadas

- Características principais:
  - Trata-se de uma estrutura sequencial, em que os elementos são organizados de forma encadeada;
  - Cada elemento (ou nó) da lista possui um campo para armazenar a informação (ou uma referência para ela) e um campo ponteiro que indica o próximo elemento da sequência.



Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Definindo um Nó:

```c
typedef struct {
    int id;
    char nome[50];
    int idade;
} Pessoa;

typedef struct No {
    Pessoa dados;
    struct No *prox;
} No;
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Definindo e inicializando uma lista encadeada simples:

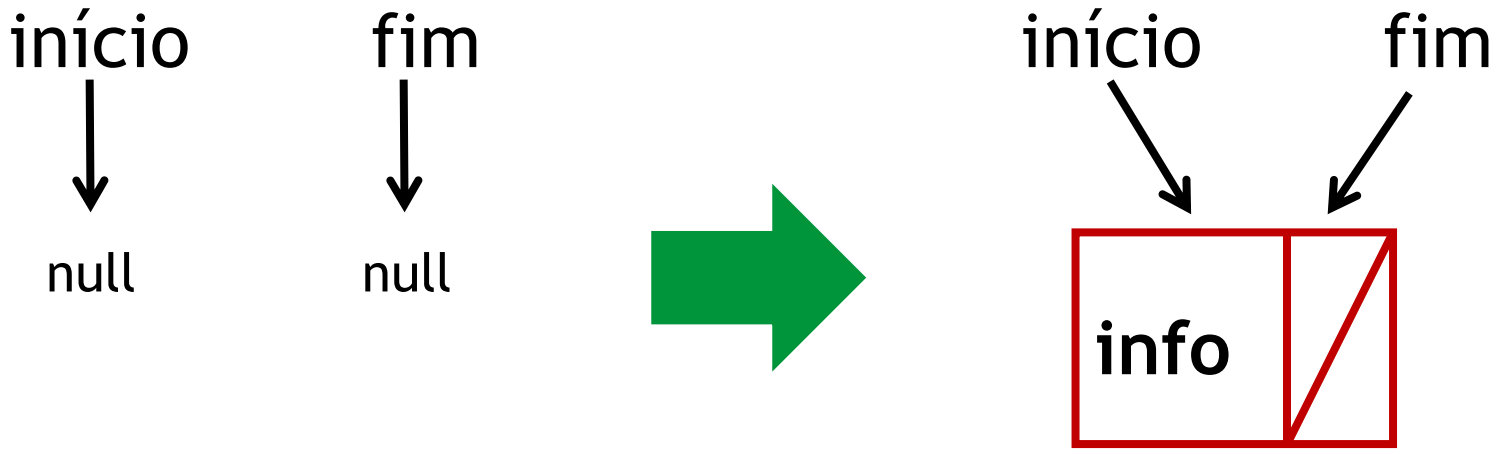
```c
typedef struct {
    No *inicio;
    No *fim;
    int tamanho;
} Lista;

void inicializarLista(Lista *lista) {
    lista->inicio = NULL;
    lista->fim = NULL;
    lista->tamanho = 0;
}
```

Listas encadeadas

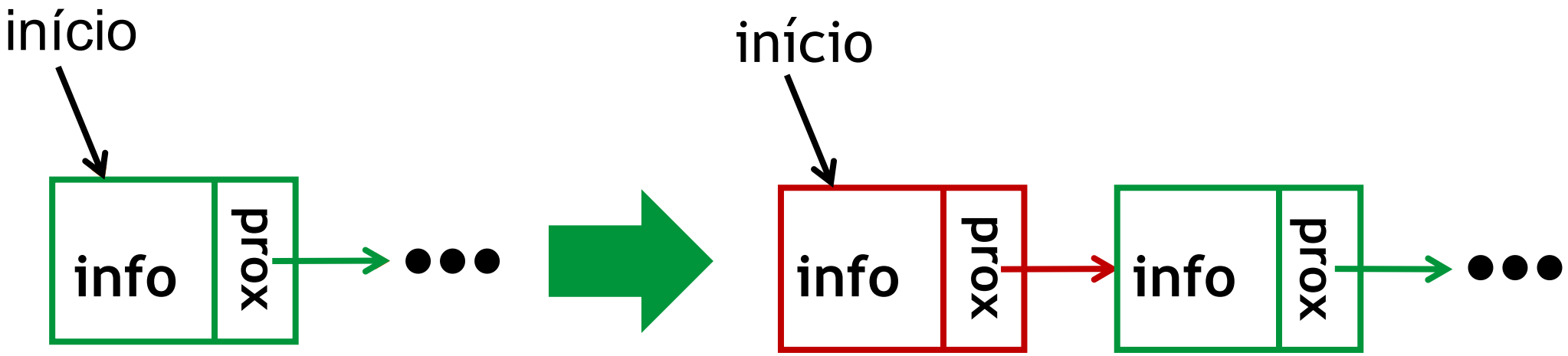
INSTITUTO FEDERAL
Baiano

- Para inserir um objeto no início da lista, deve-se criar um novo nó e fazer com que seu ponteiro “prox” referencie o atual primeiro nó da lista;
- Em seguida, o atributo “inicio” deve ser atualizado para apontar para o novo nó criado, que passa a ser o primeiro elemento da lista;
- Caso especial (lista vazia):
  - Quando a lista estiver vazia, além de atualizar “inicio”, também é necessário atualizar o atributo “fim”, fazendo com que ele aponte para o novo nó.

Listas encadeadas



Listas encadeadas



Listas encadeadas

```c
int inserirNoInicio(Lista *lista, Pessoa p) {

    No *novo = (No*) malloc(sizeof(No));
    if (novo == NULL) return 0;

    novo->dados = p;
    novo->prox = lista->inicio;

    lista->inicio = novo;

    if (lista->tamanho == 0) {
        lista->fim = novo;
    }

    lista->tamanho++;

    return 1;
}
```

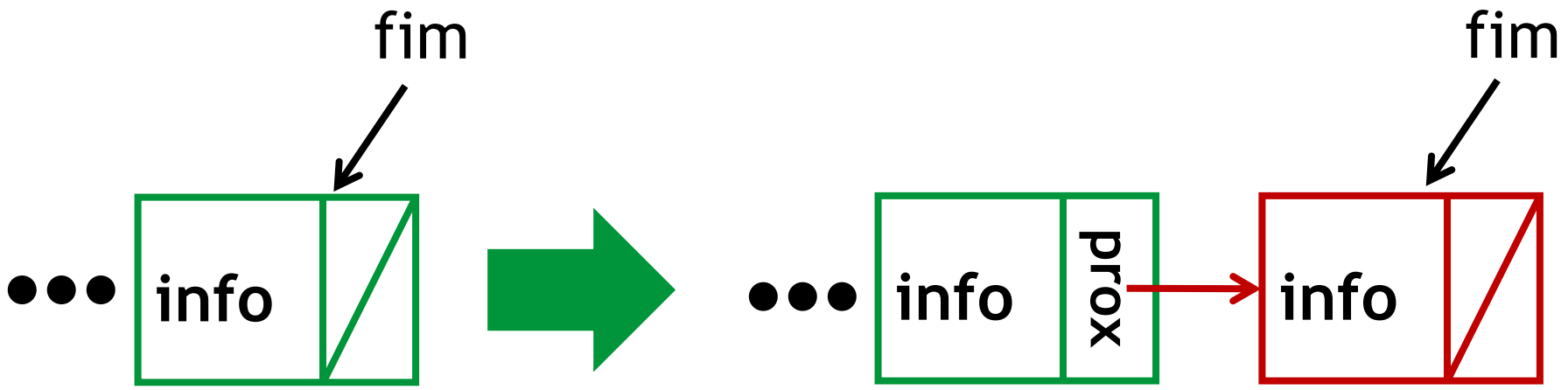
INSTITUTO FEDERAL
Baiano

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para adicionar um elemento no final da lista, caso não existisse a referência para o último nó (“fim”), seria necessário percorrer toda a lista até encontrar o último elemento e atualizar o ponteiro “prox” desse nó:
  - Em listas com muitos elementos, esse procedimento tornaria a operação lenta, pois o custo seria linear (O(n));
  - Mantendo um ponteiro para o último nó (fim), a inserção no final passa a ter custo constante (O(1)), pois não é necessário percorrer a lista.
- Caso especial (lista vazia):
  - Quando a lista estiver vazia, inserir no final ou no início produz exatamente o mesmo resultado estrutural.

Listas encadeadas



Listas encadeadas

```c
int inserirNoFim(Lista *lista, Pessoa p) {

    No *novo = (No*) malloc(sizeof(No));
    if (novo == NULL) return 0;

    novo->dados = p;
    novo->prox = NULL;

    if (lista->tamanho == 0) {
        lista->inicio = novo;
        lista->fim = novo;
    } else {
        lista->fim->prox = novo;
        lista->fim = novo;
    }

    lista->tamanho++;
    return 1;
}
```

INSTITUTO FEDERAL
Baiano

Listas encadeadas

- Percorrendo e imprimindo todos os elementos:

```c
void imprimirLista(Lista *lista) {

    if (lista->inicio == NULL) {
        printf("Lista vazia.\n");
        return;
    }

    No *atual = lista->inicio;
    int posicao = 0;

    while (atual != NULL) {
        printf("Posicao %d -> ID: %d | Nome: %s | Idade: %d\n",
               posicao,
               atual->dados.id,
               atual->dados.nome,
               atual->dados.idade);

        atual = atual->prox;
        posicao++;
    }
}
```

INSTITUTO FEDERAL
Baiano

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para inserir um elemento em uma posição específica da lista, é necessário obter a referência do nó anterior à posição desejada, pois será ele que terá seu ponteiro “prox” atualizado;
- Para facilitar esse processo, pode-se implementar um método auxiliar responsável por retornar o nó correspondente a uma determinada posição;
- Esse método deve, obrigatoriamente, verificar se a posição informada é válida antes de realizar a busca.

Quando a posição é válida?

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para inserir um elemento em uma posição específica da lista, é necessário obter a referência do nó anterior à posição desejada, pois será ele que terá seu ponteiro “prox” atualizado;
- Para facilitar esse processo, pode-se implementar um método auxiliar responsável por retornar o nó correspondente a uma determinada posição;
- Esse método deve, obrigatoriamente, verificar se a posição informada é válida antes de realizar a busca.

Quando a posição é válida?

  - A posição será válida quando for maior ou igual a zero e menor ou igual ao tamanho atual da lista.

Listas encadeadas

INSTITUTO FEDERAL
Baiano

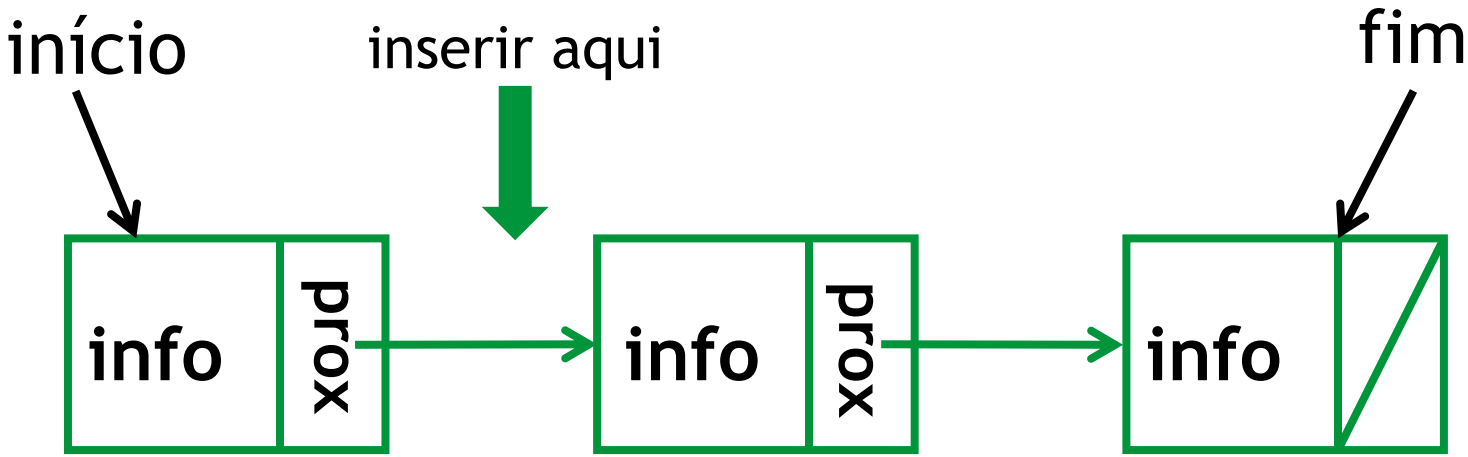
```c
No* obterNoAnterior(Lista *lista, int posicao) {

    if (posicao <= 0 || posicao > lista->tamanho) {
        return NULL;
    }

    No *atual = lista->inicio;

    for (int i = 0; i < posicao - 1; i++) {
        atual = atual->prox;
    }

    return atual;
}
```

Listas encadeadas

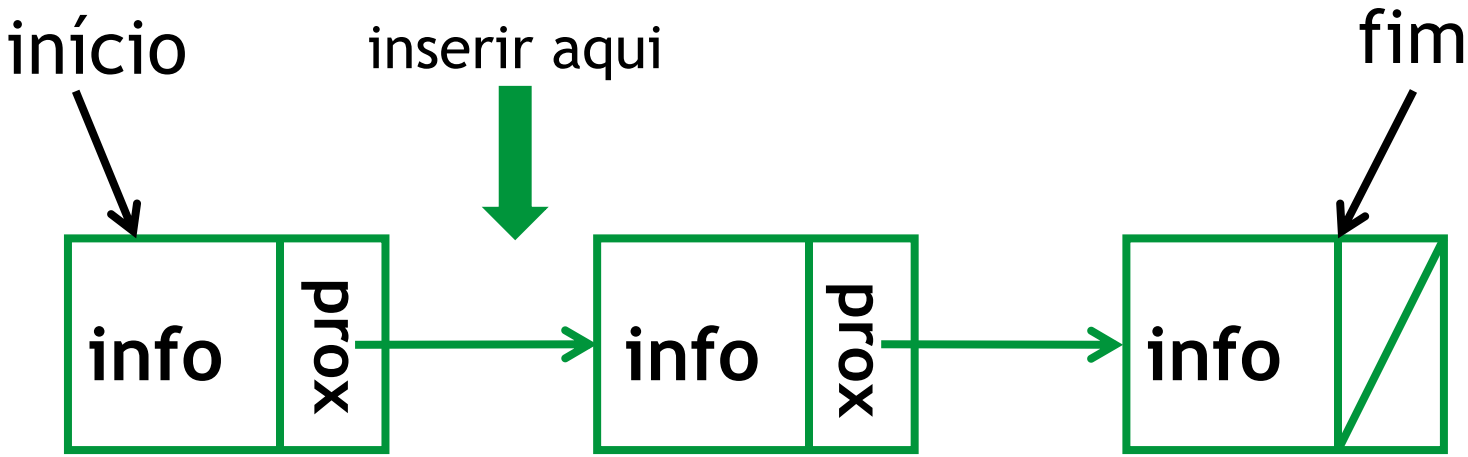
INSTITUTO FEDERAL
Baiano

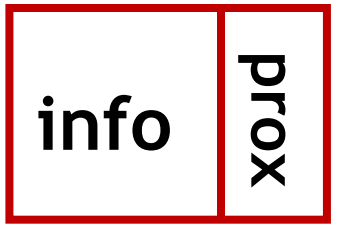
- É importante observar que o método obterNoAnterior(Lista *lista, int posicao) possui custo linear (O(n)), pois pode ser necessário percorrer a lista desde o início até a posição desejada;
- Com a implementação de obterNoAnterior, o método adicionarPorPosicao (Lista *lista, int posicao) torna-se simples de implementar, basta:
  - O nó anterior deve passar a apontar para o novo nó;
  - O novo nó deve apontar para o antigo prox do nó anterior;
- Casos especiais: posição == 0 ou posição == tamanho da lista.

Listas encadeadas

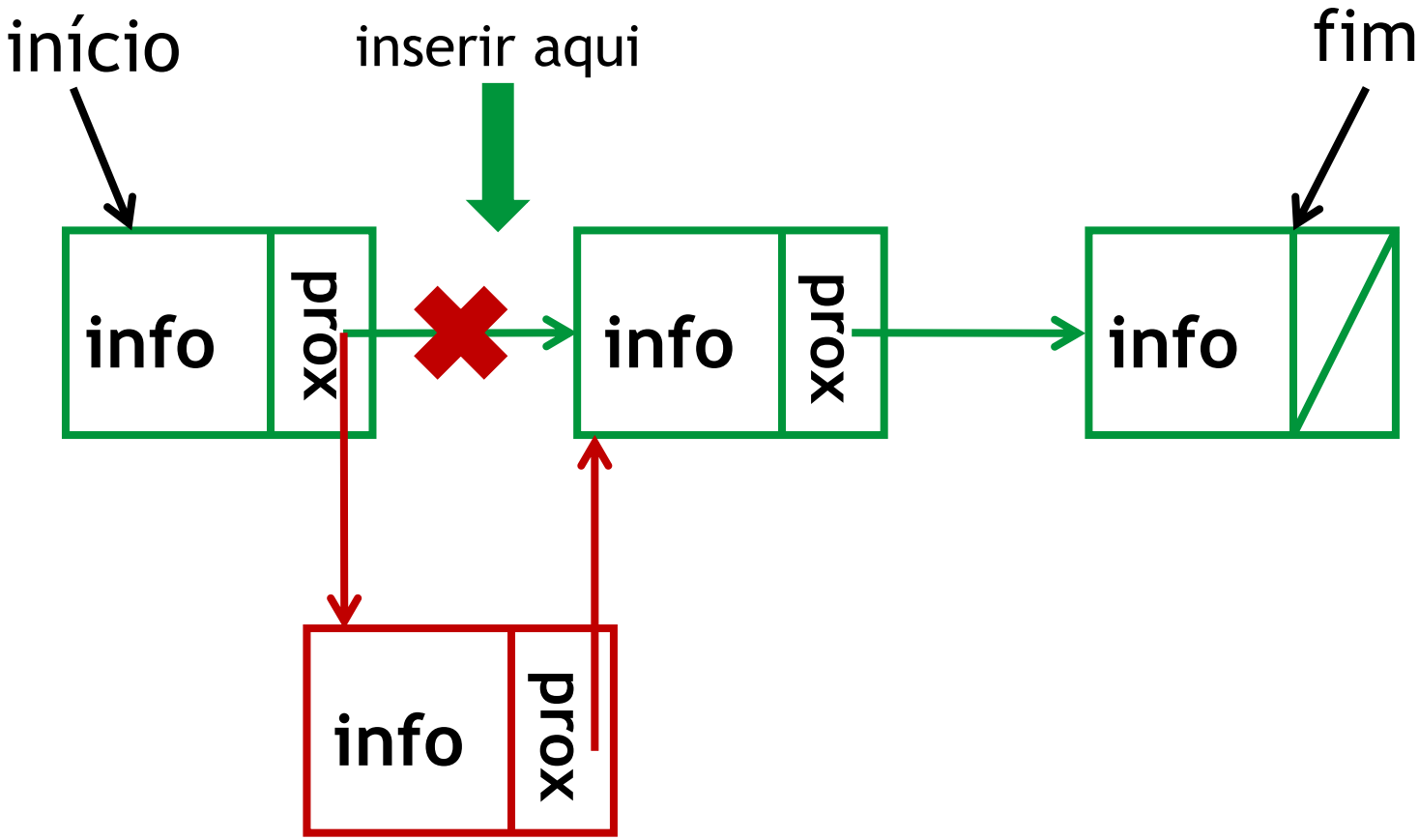


Listas encadeadas





Listas encadeadas



Listas encadeadas

```c
int adicionarPorPosicao(Lista *lista, Pessoa p, int posicao) {

    if (posicao < 0 || posicao > lista->tamanho)
        return 0;

    if (posicao == 0)
        return inserirNoInicio(lista, p);

    if (posicao == lista->tamanho)
        return inserirNoFim(lista, p);

    No *anterior = obterNoAnterior(lista, posicao);
    if (anterior == NULL)
        return 0;

    No *novo = (No*) malloc(sizeof(No));
    if (novo == NULL)
        return 0;

    novo->dados = p;

    novo->prox = anterior->prox;
    anterior->prox = novo;

    lista->tamanho++;

    return 1;
}
```

INSTITUTO FEDERAL
Baiano

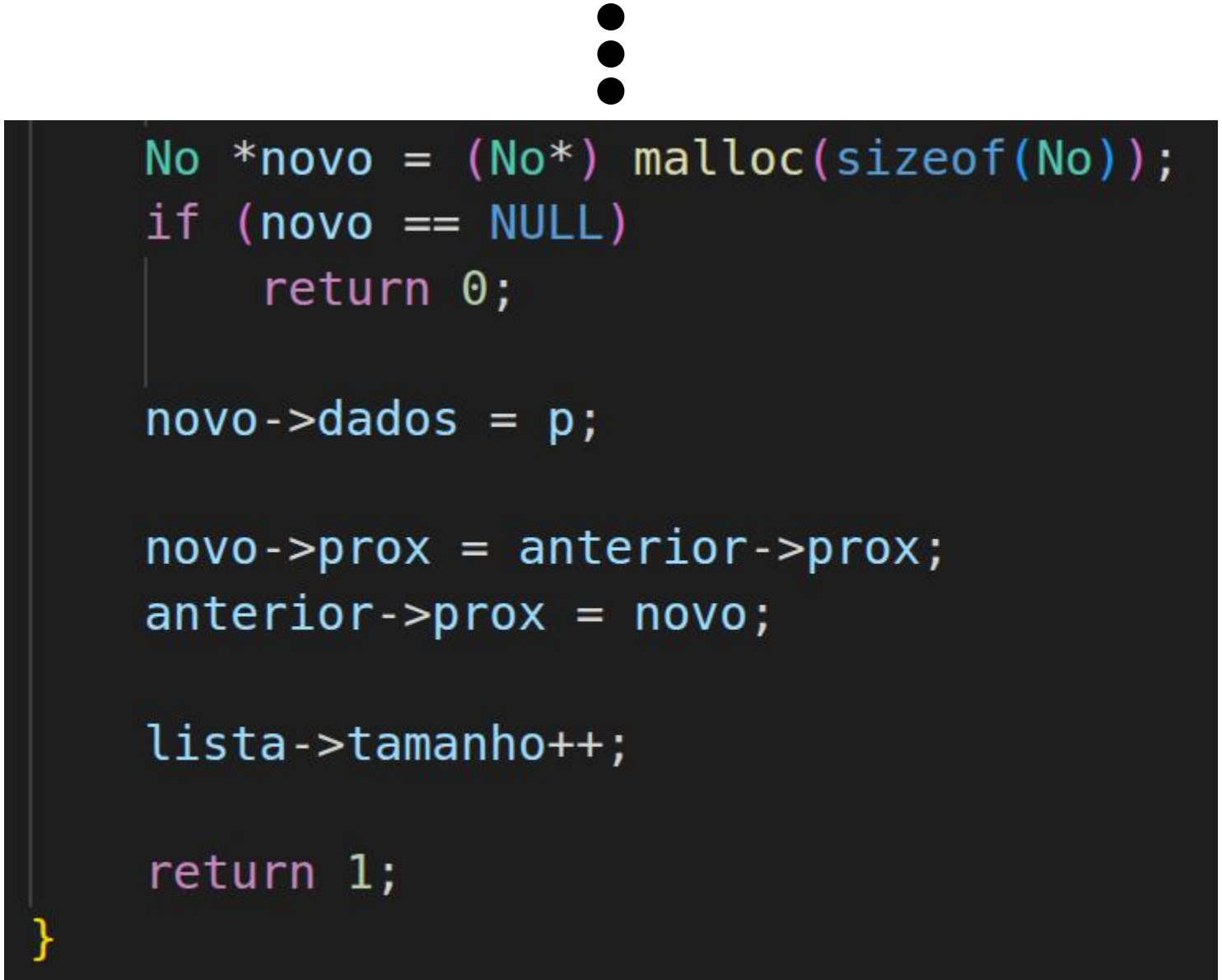
Listas encadeadas

INSTITUTO FEDERAL
Baiano

```
int adicionarPorPosicao(Lista *lista, Pessoa p, int posicao) {

    if (posicao < 0 || posicao > lista->tamanho)
        return 0;

    if (posicao == 0)
        return inserirNoInicio(lista, p);

    if (posicao == lista->tamanho)
        return inserirNoFim(lista, p);

    No *anterior = obterNoAnterior(lista, posicao);
    if (anterior == NULL)
        return 0;
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano



```
    No *novo = (No*) malloc(sizeof(No));
    if (novo == NULL)
        return 0;

    novo->dados = p;

    novo->prox = anterior->prox;
    anterior->prox = novo;

    lista->tamanho++;

    return 1;
}
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para acessar o tamanho da lista, basta retornar o valor armazenado no atributo tamanho;
- Como o tamanho é mantido e atualizado a cada inserção ou remoção, não é necessário percorrer a lista para contabilizar os elementos:
  - O consumo de tempo da operação é constante (O(1)).

```
int obterTamanho(Lista *lista) {
    return lista->tamanho;
}
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para verificar se um objeto está presente na lista, é necessário percorrer a estrutura desde o primeiro nó, comparando o objeto informado com cada elemento armazenado;
- Caso algum elemento satisfaça o critério de comparação (por exemplo, mesmo id ou mesmo nome), a busca pode ser encerrada e o resultado retornado;
- Se nenhum elemento correspondente for encontrado após a travessia completa da lista, conclui-se que o objeto não está presente;
- Como pode ser necessário percorrer todos os nós, o custo dessa operação é linear (O(n)).

Listas encadeadas

INSTITUTO FEDERAL
Baiano

```c
int buscarPorNome(Lista *lista, const char *nome) {

    No *atual = lista->inicio;

    while (atual != NULL) {

        if (strcmp(atual->dados.nome, nome) == 0) {
            return 1;
        }

        atual = atual->prox;
    }

    return 0;
}
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para recuperar um objeto em uma determinada posição, é necessário localizar o nó correspondente a essa posição e, em seguida, acessar seus dados;
- Para isso, pode-se utilizar o método buscarNoAnterior(Lista *lista, int posicao), que permite obter o nó anterior à posição desejada;
  - A partir do nó anterior, basta acessar seu ponteiro prox para alcançar o nó onde o objeto está armazenado;
- É importante destacar que esse procedimento possui custo linear (O(n)).
  - Essa é uma das principais desvantagens da lista encadeada em comparação com a implementação baseada em vetor

Listas encadeadas

INSTITUTO FEDERAL
Baiano

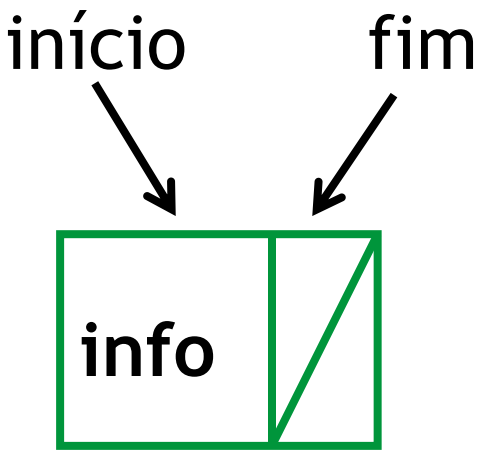
```
Pessoa recuperarPorPosicao(Lista *lista, int posicao) {

    Pessoa vazio = {0, "", 0};

    if (posicao < 0 || posicao >= lista->tamanho)
        return vazio;

    if (posicao == 0)
        return lista->inicio->dados;

    No *anterior = buscarNoAnterior(lista, posicao);
    if (anterior == NULL || anterior->prox == NULL)
        return vazio;

    return anterior->prox->dados;
}
```

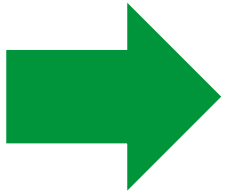
Listas encadeadas

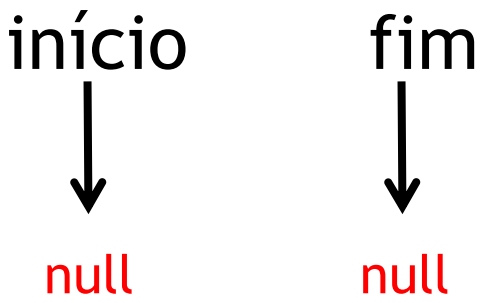
INSTITUTO FEDERAL
Baiano

- Para remover um elemento do início, é necessário verificar se a lista não está vazia, ou seja, se a operação é válida;
- Em seguida, deve-se atualizar a referência que aponta para o primeiro nó (inicio), fazendo-a avançar para o próximo nó da lista;
- Após essa atualização, é fundamental verificar se a lista ficou vazia;
- Caso isso ocorra, também é necessário atualizar a referência fim, atribuindo-lhe NULL, a fim de evitar um estado inconsistente na estrutura.

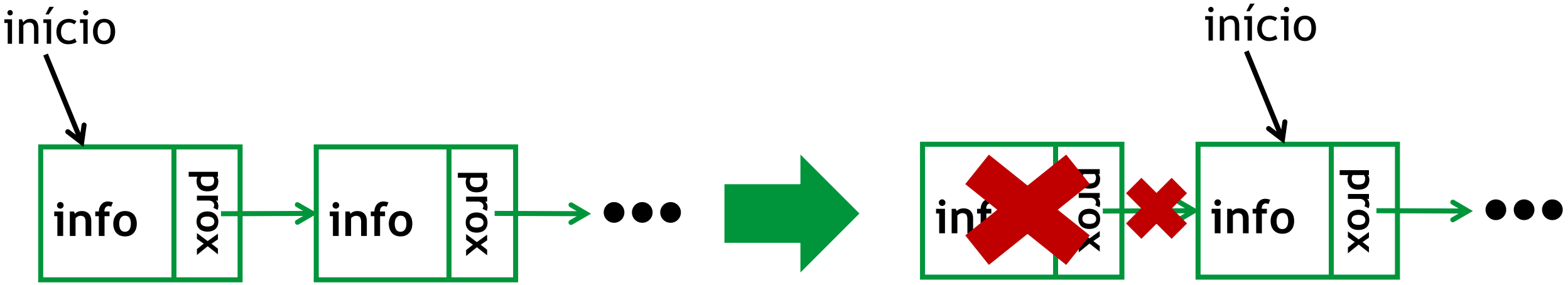
Listas encadeadas







Listas encadeadas



Listas encadeadas

```c
int removerNoInicio(Lista *lista) {

    if (lista->inicio == NULL)
        return 0;

    No *remover = lista->inicio;

    lista->inicio = remover->prox;

    if (lista->inicio == NULL) {
        lista->fim = NULL;
    }

    free(remover);
    lista->tamanho--;

    return 1;
}
```

INSTITUTO FEDERAL
Baiano

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Para remover o último elemento da lista, é necessário, inicialmente, verificar se a lista não está vazia;
- Caso geral = lista com mais de um elemento:
  - É necessário localizar o penúltimo nó, o ponteiro prox do penúltimo nó deve ser atualizado para NULL;
  - Em seguida, a referência fim deve passar a apontar para o penúltimo nó
- Caso especial = lista com apenas um elemento:
  - Remover do final é equivalente a remover do início;
  - Nesse caso, tanto inicio quanto fim devem ser atualizados para NULL;
- Como é preciso percorrer a lista para encontrar o penúltimo nó, essa operação possui custo linear (O(n)) em listas simplesmente encadeadas.

Listas encadeadas

```c
int removerNoFim(Lista *lista) {

    if (lista->inicio == NULL)
        return 0;

    if (lista->inicio == lista->fim) {
        free(lista->inicio);
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho--;
        return 1;
    }

    No *atual = lista->inicio;

    while (atual->prox != lista->fim) {
        atual = atual->prox;
    }

    free(lista->fim);
    atual->prox = NULL;
    lista->fim = atual;

    lista->tamanho--;

    return 1;
}
```

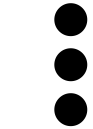
INSTITUTO FEDERAL
Baiano

Listas encadeadas

INSTITUTO FEDERAL
Baiano

```c
int removerNoFim(Lista *lista) {

    if (lista->inicio == NULL)
        return 0;

    if (lista->inicio == lista->fim) {
        free(lista->inicio);
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho--;
        return 1;
    }
```

Listas encadeadas



```
    No *atual = lista->inicio;

    while (atual->prox != lista->fim) {
        atual = atual->prox;
    }

    free(lista->fim);
    atual->prox = NULL;
    lista->fim = atual;

    lista->tamanho--;

    return 1;
}
```

INSTITUTO FEDERAL
Baiano

Listas encadeadas

- Como remover o último elemento de forma eficiente?

Listas encadeadas

- Como remover o último elemento de forma eficiente?

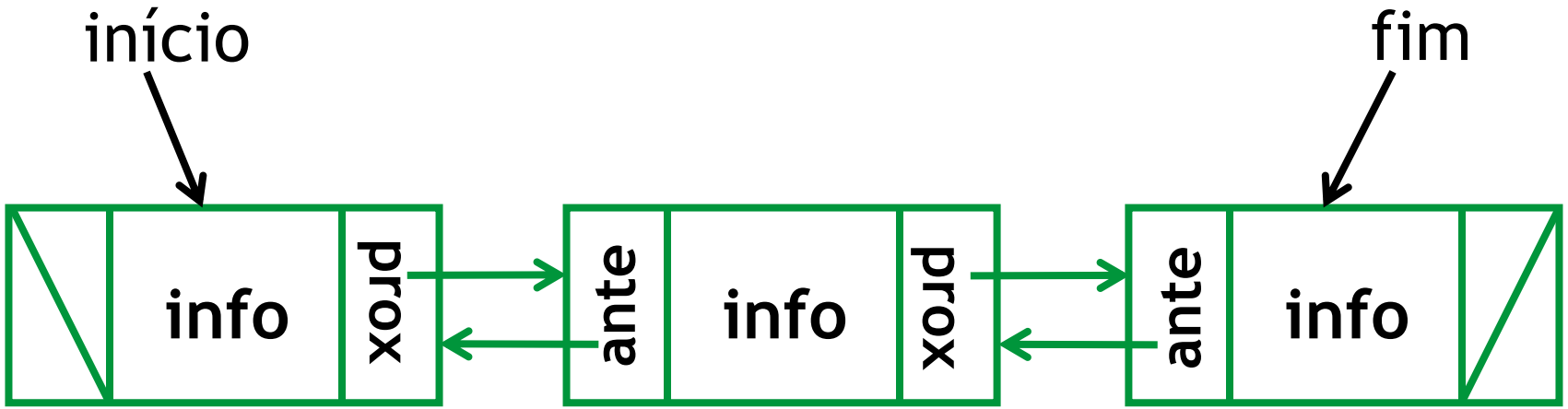
Listas encadeadas duplas

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Podemos implementar uma lista duplamente encadeada, onde cada nó mantém duas referências:
  - uma para o próximo nó da sequência;
  - outra para o nó anterior;
- Dessa forma, a estrutura permite navegação em ambos os sentidos, o que pode tornar determinadas operações mais eficientes;
- Em contrapartida, há um pequeno aumento no consumo de memória, pois cada nó passa a armazenar dois ponteiros em vez de apenas um.

Listas encadeadas

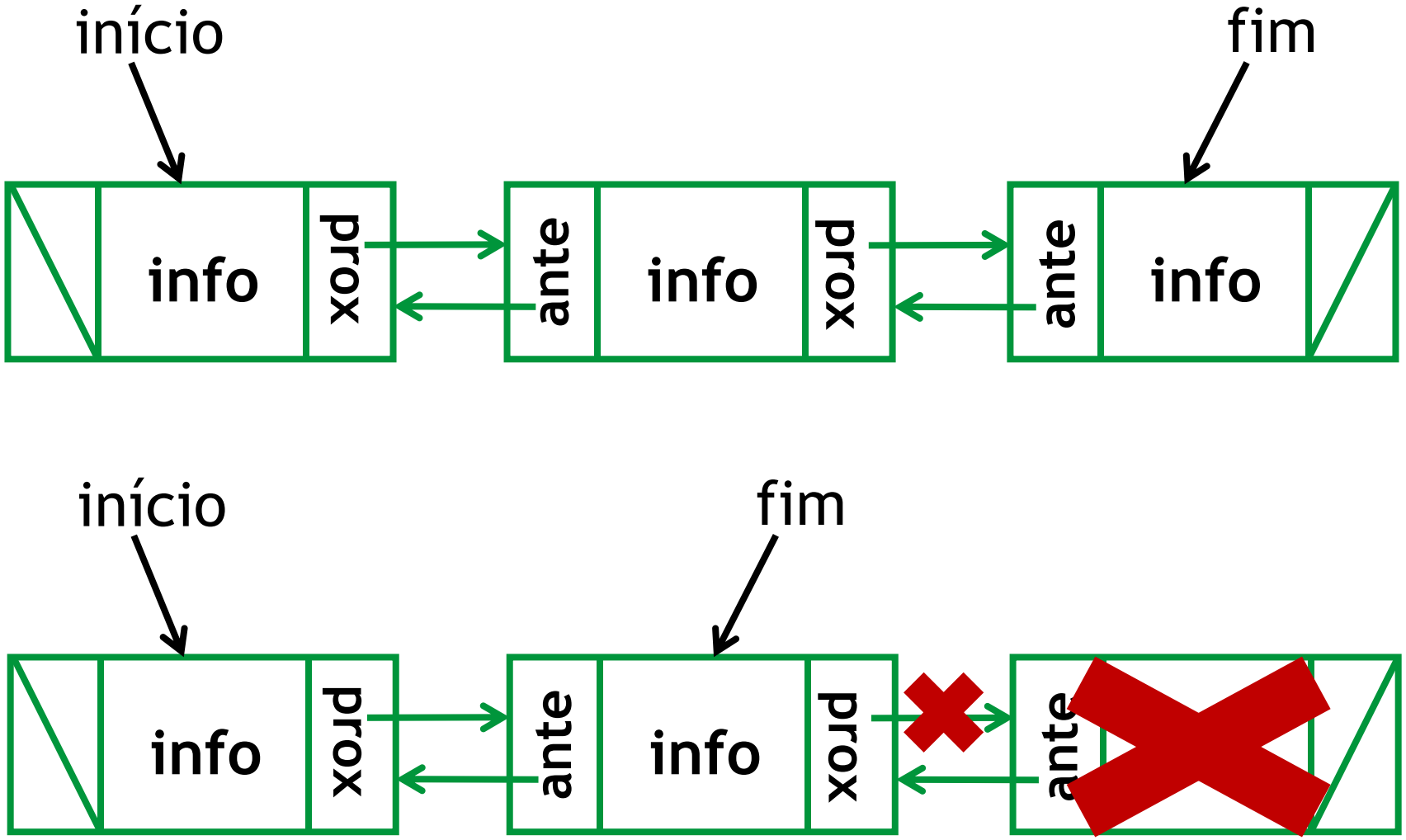


Listas encadeadas

INSTITUTO FEDERAL
Baiano

```c
typedef struct No {
    Pessoa dados;
    struct No *prox;
    struct No *ant;
} No;

typedef struct {
    No *inicio;
    No *fim;
    int tamanho;
} ListaDupla;
```

Listas encadeadas



Listas encadeadas

```c
int removerNoFim(ListaDupla *lista) {

    if (lista->fim == NULL)
        return 0;

    if (lista->inicio == lista->fim) {
        free(lista->fim);
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho--;
        return 1;
    }

    No *remover = lista->fim;

    lista->fim = remover->ant;
    lista->fim->prox = NULL;

    free(remover);
    lista->tamanho--;

    return 1;
}
```

INSTITUTO FEDERAL
Baiano

Listas encadeadas

INSTITUTO FEDERAL
Baiano

```c
int removerNoFim(ListaDupla *lista) {

    if (lista->fim == NULL)
        return 0;

    if (lista->inicio == lista->fim) {
        free(lista->fim);
        lista->inicio = NULL;
        lista->fim = NULL;
        lista->tamanho--;
        return 1;
    }
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

```
    No *remover = lista->fim;

    lista->fim = remover->ant;
    lista->fim->prox = NULL;

    free(remover);
    lista->tamanho--;

    return 1;
}
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

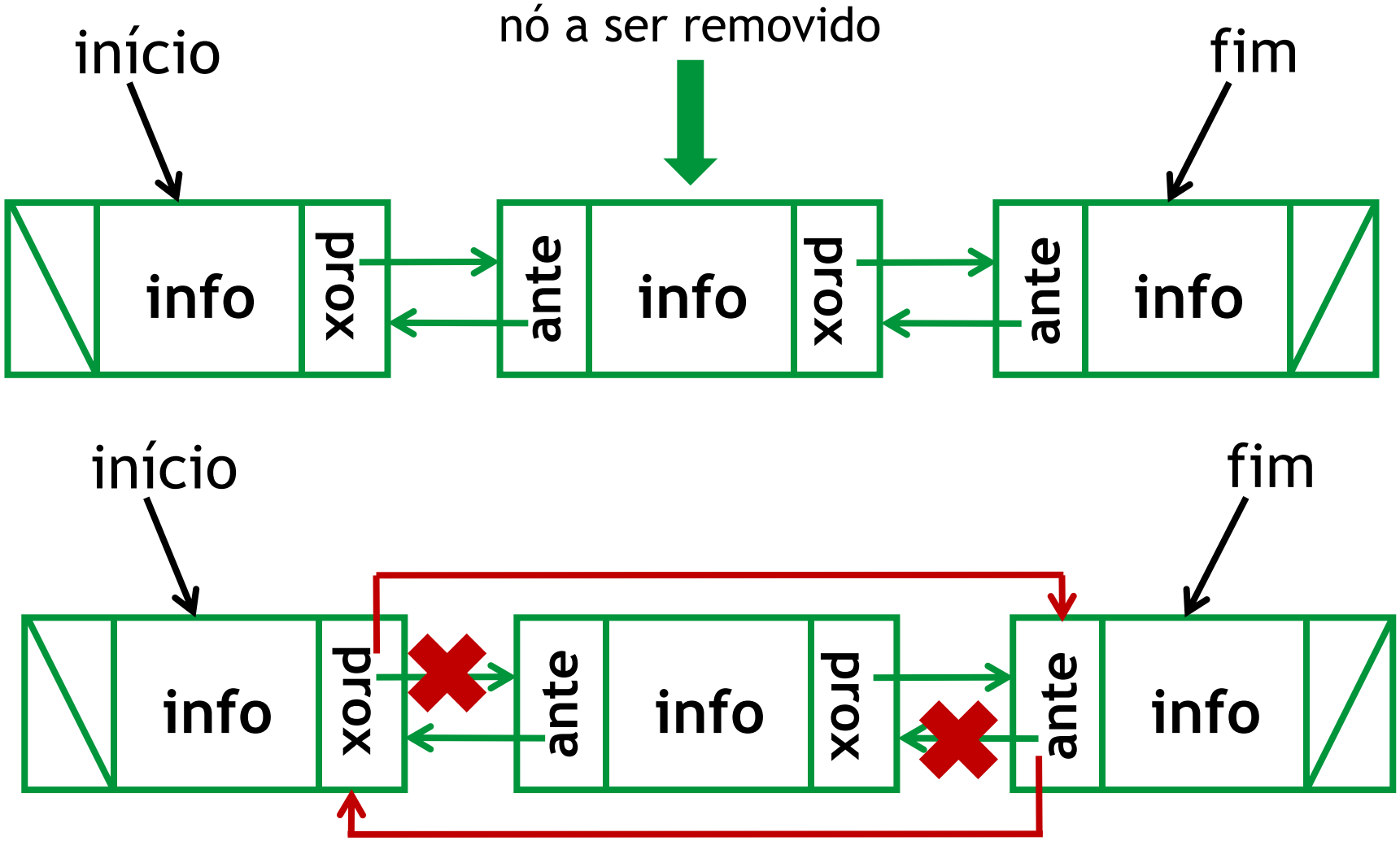
- Para remover um elemento de uma posição específica, inicialmente, deve-se verificar se a posição informada é válida (0 ≤ posição < tamanho da lista);
- Em seguida, é necessário identificar se a remoção ocorre em um dos casos particulares:
  - Remoção no início da lista;
  - Remoção no final da lista.

Listas encadeadas

INSTITUTO FEDERAL
Baiano

- Caso a remoção seja em uma posição intermediária, é preciso:
  - Localizar o nó correspondente à posição informada;
  - Atualizar a referência do nó anterior para que ele passe a apontar para o próximo nó;
  - Atualizar a referência do nó seguinte para que ele passe a apontar para o nó anterior;
  - Por fim, liberar a memória do nó removido e atualizar o tamanho da lista.

Listas encadeadas



Listas encadeadas

```c
int removerPorPosicao(ListaDupla *lista, int posicao) {

    if (posicao < 0 || posicao >= lista->tamanho)
        return 0;

    if (posicao == 0)
        return removerNoInicio(lista);

    if (posicao == lista->tamanho - 1)
        return removerNoFim(lista);

    No *atual = lista->inicio;

    for (int i = 0; i < posicao; i++) {
        atual = atual->prox;
    }

    atual->ant->prox = atual->prox;
    atual->prox->ant = atual->ant;

    free(atual);
    lista->tamanho--;

    return 1;
}
```

INSTITUTO FEDERAL
Baiano

Listas encadeadas

INSTITUTO FEDERAL
Baiano

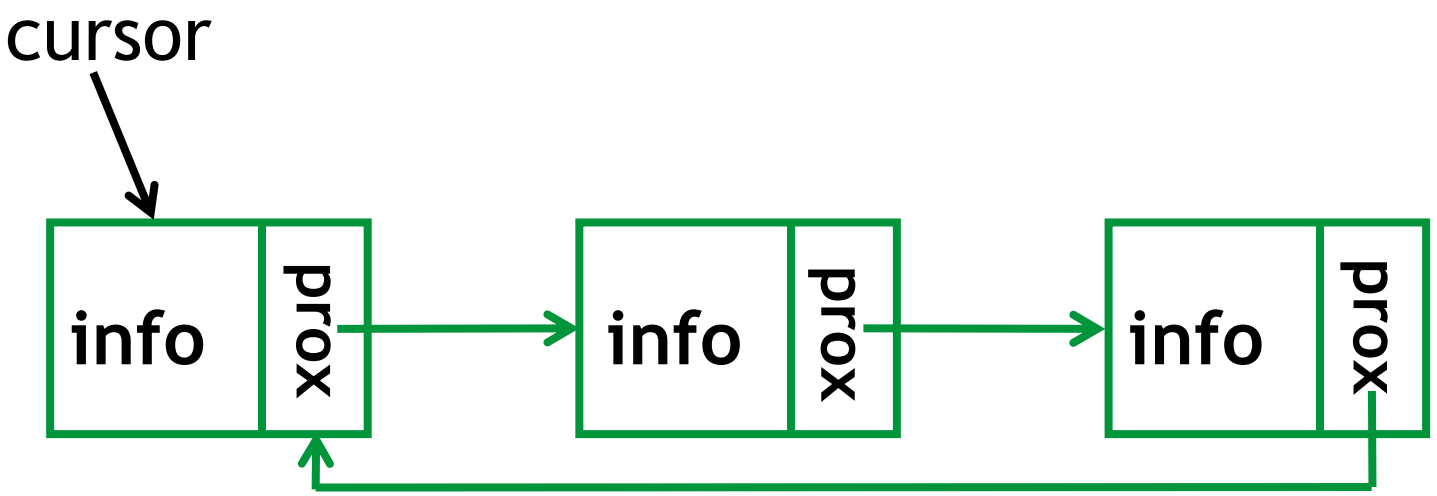
```
int removerPorPosicao(ListaDupla *lista, int posicao) {

    if (posicao < 0 || posicao >= lista->tamanho)
        return 0;

    if (posicao == 0)
        return removerNoInicio(lista);

    if (posicao == lista->tamanho - 1)
        return removerNoFim(lista);
```

Listas encadeadas

INSTITUTO FEDERAL
Baiano

```
    No *atual = lista->inicio;

    for (int i = 0; i < posicao; i++) {
        atual = atual->prox;
    }

    atual->ant->prox = atual->prox;
    atual->prox->ant = atual->ant;

    free(atual);
    lista->tamanho--;

    return 1;
}
```

Listas circulares



Exercícios

INSTITUTO FEDERAL
Baiano

- Implemente todas as funcionalidades da estrutura de dados Lista Encadeada, considerando:
  - Lista simplesmente encadeada
  - Lista duplamente encadeada
- Para cada tipo de Listas implemente as funções:
  - Inicialização da lista
  - Inserção no início, final e em uma posição específica
  - Remoção no início, final e em uma posição específica
  - Busca de elemento
  - Recuperação por posição
  - Impressão dos elementos
  - Consulta do tamanho

Dúvidas

INSTITUTO FEDERAL
Baiano

ESTRUTURA DE DADOS

Curso de Licenciatura em Ciências da Computação

Lucas Sampaio Leite