Programação de Computadores II

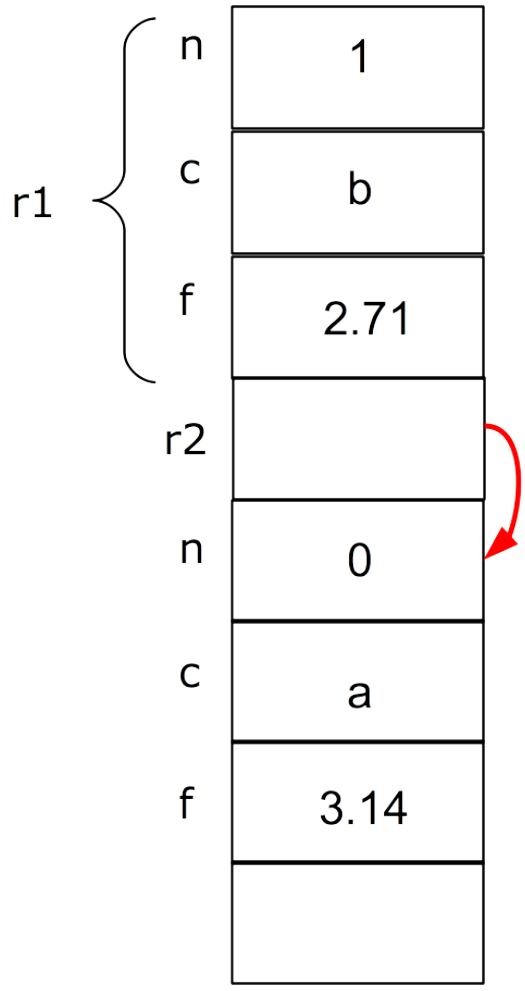
Lista Encadeada/Ligada

Raphael M.

Slides adaptados do material de Karina Mochetti

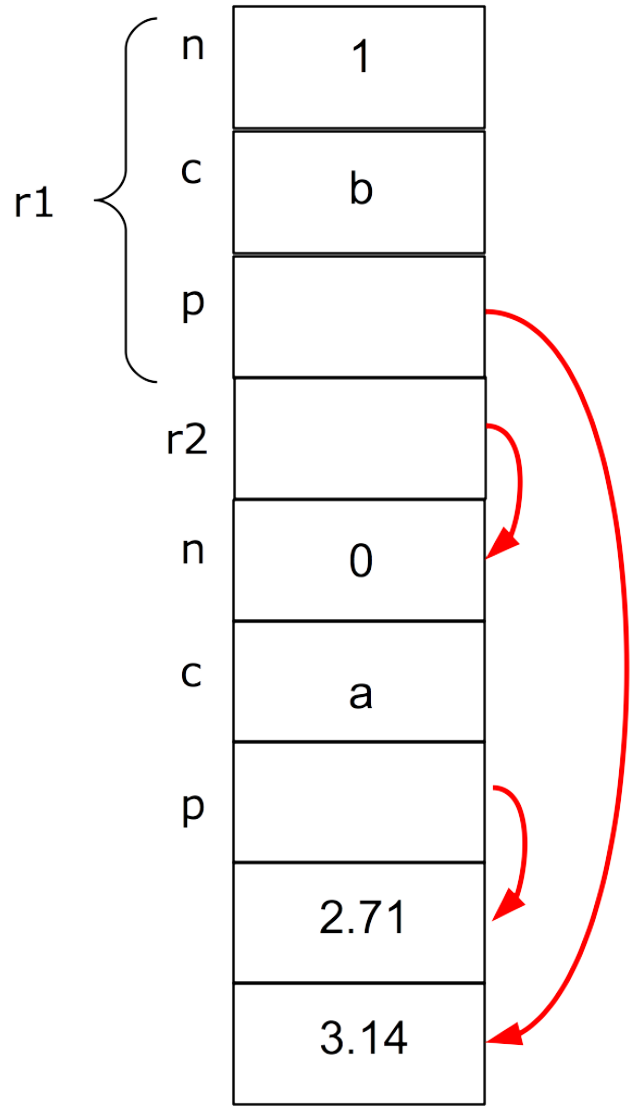
Ponteiro para Registro

```
typedef struct reg
     {   int  n;
     char c;
     float
     f;
}  Reg;

int  main ()
     {   Reg
     r1;   Reg
     *r2;
     r2 = (Reg *)malloc(sizeof(Reg));
     r2->n = 0;
     r2->c = 'a';
     r2->f =
     3.14;
     r1.n = 1;
     r1.c =
     'b';
     r1.f =
     2.71;
     return 0;
}
```



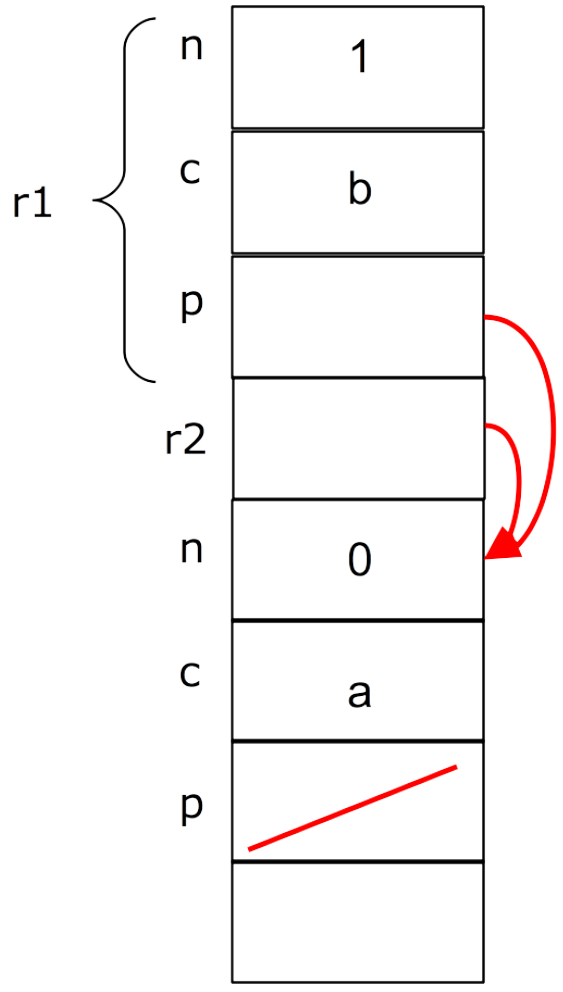
Ponteiro em Registro

```
typedef struct reg
     {   int  n;
     char c;
     float
     *p;
}  Reg;

int  main ()
     {   Reg
     r1;   Reg
     *r2;
     r2 = (Reg *)malloc(sizeof(Reg));
     r2->n = 0;
     r2->c = 'a';
     r2->p = (float
     *)malloc(sizeof(float));
     *(r2->p) = 2.71;
     r1.n = 1;
     r1.c = 'b';
     r1.p = (float
     *)malloc(sizeof(float));
     *(r1.p) = 3.14;
     return 0;
}
```



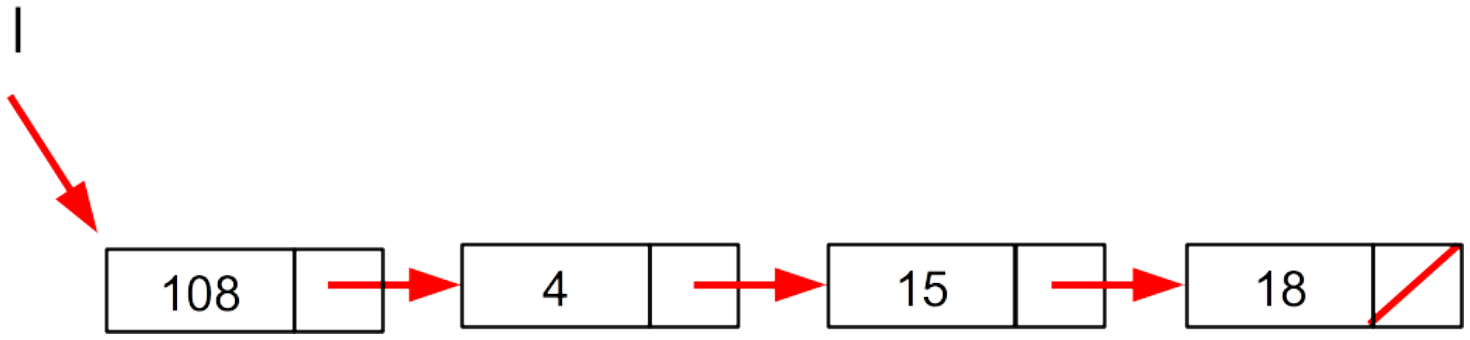
Ponteiro em Registro para o Registro

```
typedef struct reg {
    int n;
    char c;
    struct reg *p;
} Reg;

int main () {
    Reg r1;
    Reg *r2;
    r2 = (Reg *)malloc(sizeof(Reg));
    r2->n = 0;
    r2->c = 'a';
    r2->p = NULL;
    r1.n = 1;
    r1.c = 'b';
    r1.p = r2;
    return 0;
}
```



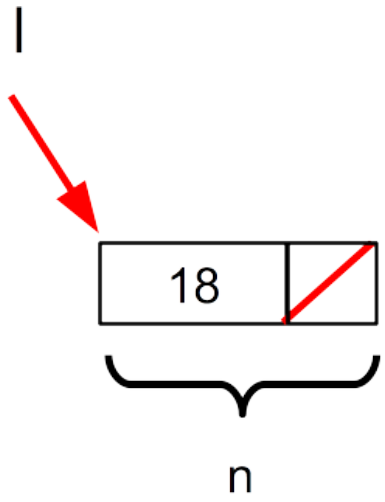
Lista Ligada

Uma lista ligada é uma estrutura de dados linear composta por células que apontam para o próximo elemento da lista. Cada cédula é normalmente chamada de nó.



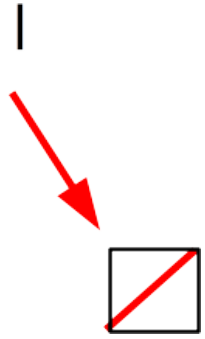
Lista Ligada

Cada cédula da lista é um nó e possui o dado principal e um ponteiro para o próximo nó. A lista é dada por um ponteiro para o primeiro nó.



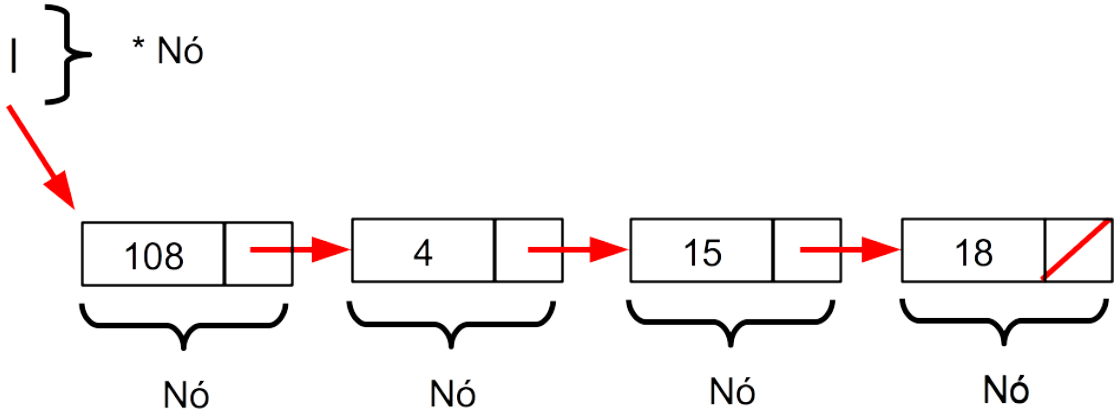
Lista Ligada

O último elemento de uma lista sempre aponta para NULL. Uma lista vazia é dada, portanto, por um ponteiro para NULL.



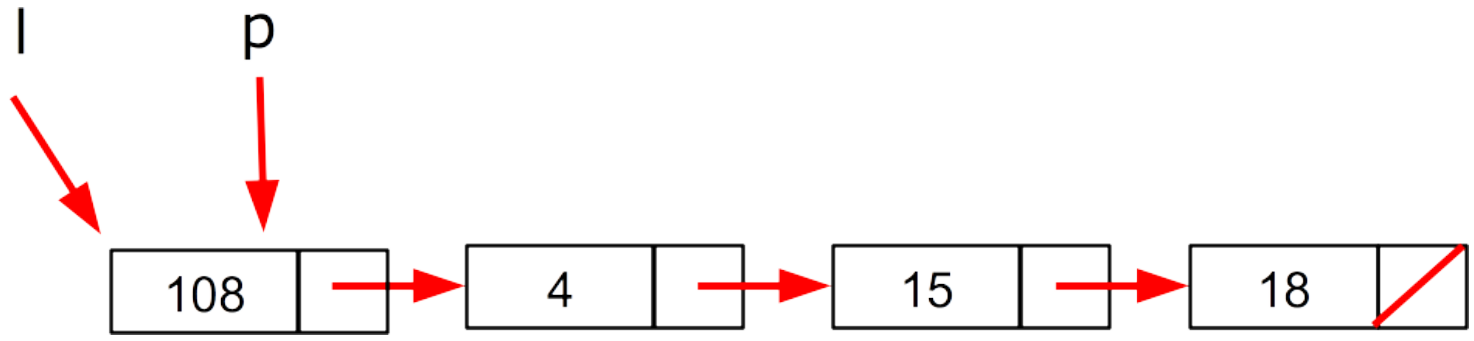
Lista Ligada: Declaração

```
typedef struct no {
    int dado;
    struct no* prox;
} No;

No *l = NULL;
No *n;

n = (No *) malloc(sizeof(No));
n->dado = 108;
n->prox = NULL;
l = n;
```



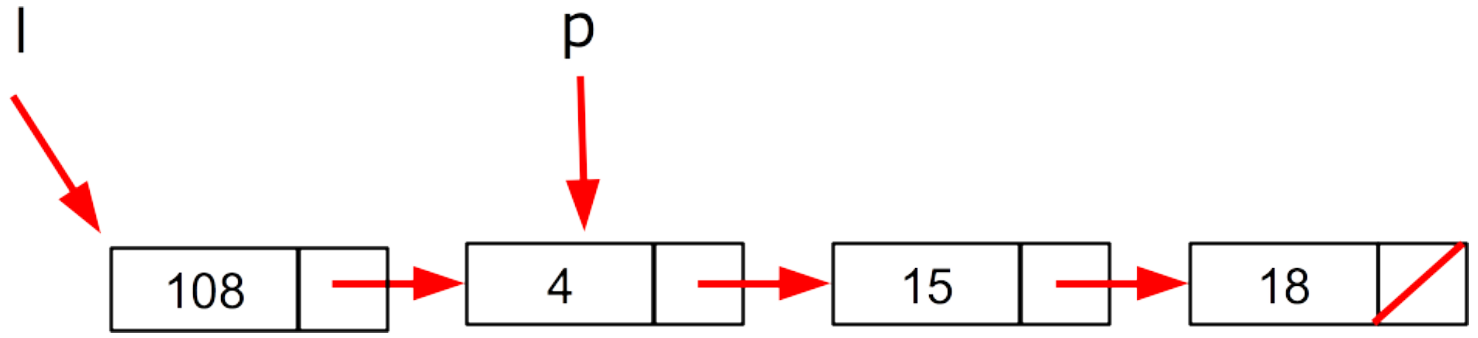
Lista Ligada: Percorrer

```
typedef struct no {
    int dado;
    struct no* prox;
} No;

void percorreLista(No* l) {
    No *p = l;
    while (p != NULL)
        p = p->prox;
}
```



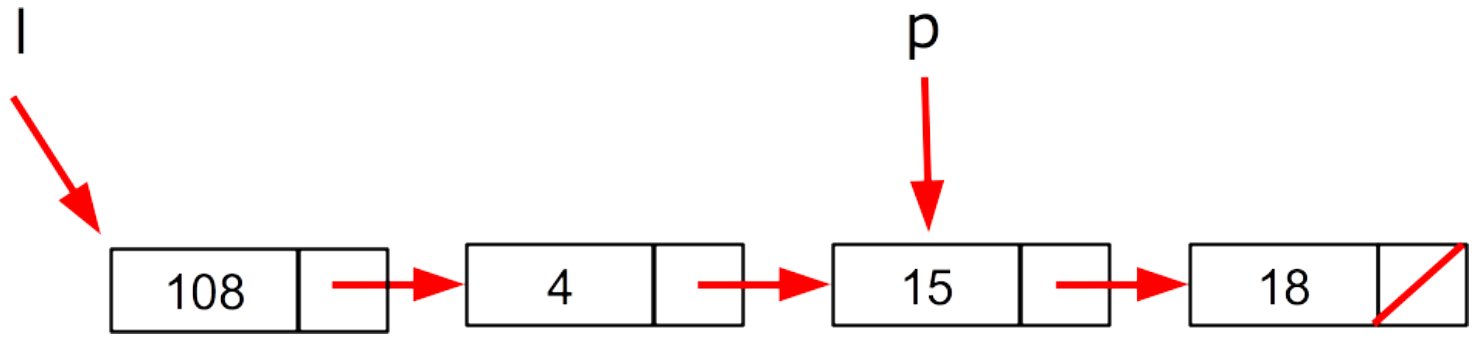
Lista Ligada: Percorrer

```
typedef struct no {
    int dado;
    struct no* prox;
} No;

void percorreLista(No* l) {
    No *p = l;
    while (p != NULL)
        p = p->prox;
}
```



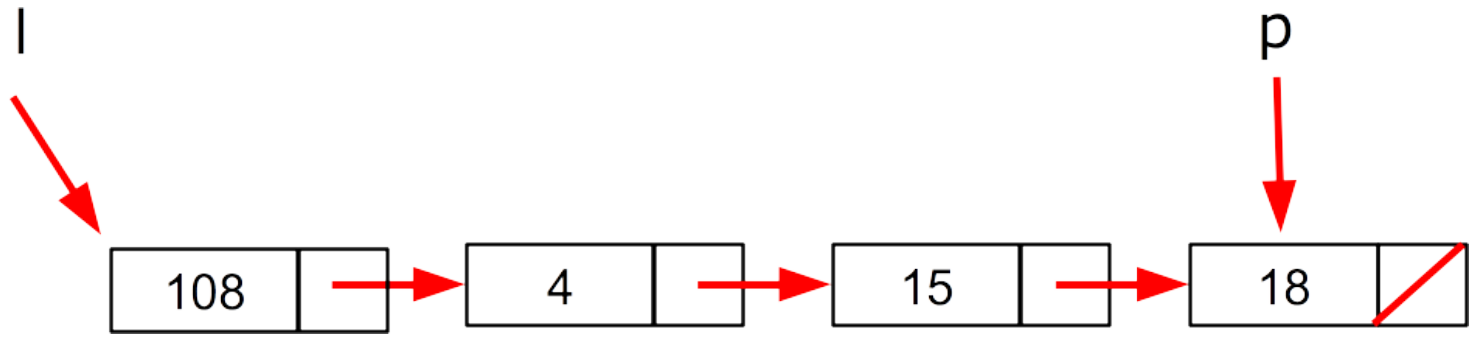
Lista Ligada: Percorrer

```
typedef struct no {
    int dado;
    struct no* prox;
} No;

void percorreLista(No* l) {
    No *p = l;
    while (p != NULL)
        p = p->prox;
}
```



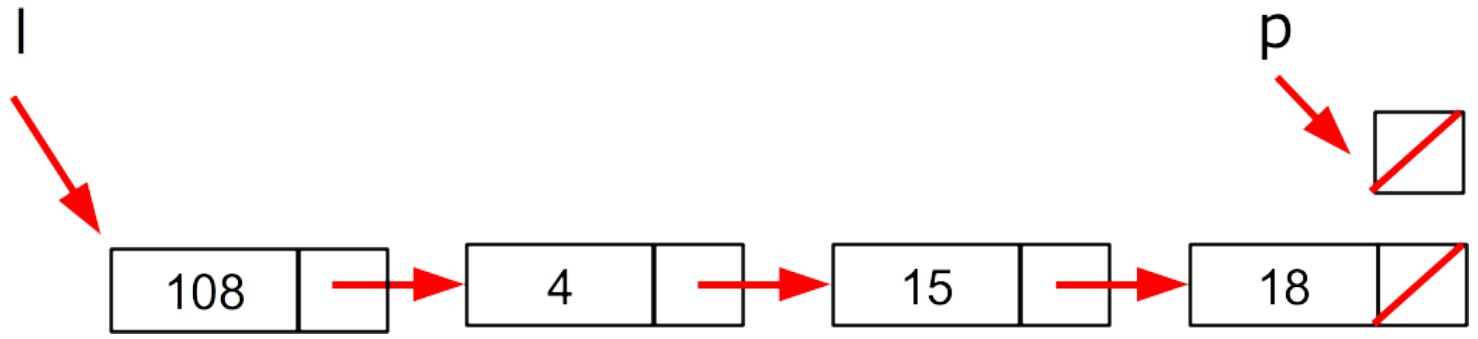
Lista Ligada: Percorrer

```
typedef struct no {
    int dado;
    struct no* prox;
} No;

void percorreLista(No* l) {
    No *p = l;
    while (p != NULL)
        p = p->prox;
}
```



Lista Ligada: Percorrer

```
typedef struct no {
    int dado;
    struct no* prox;
} No;

void percorreLista(No* l) {
    No *p = l;
    while (p != NULL)
        p = p->prox;
}
```

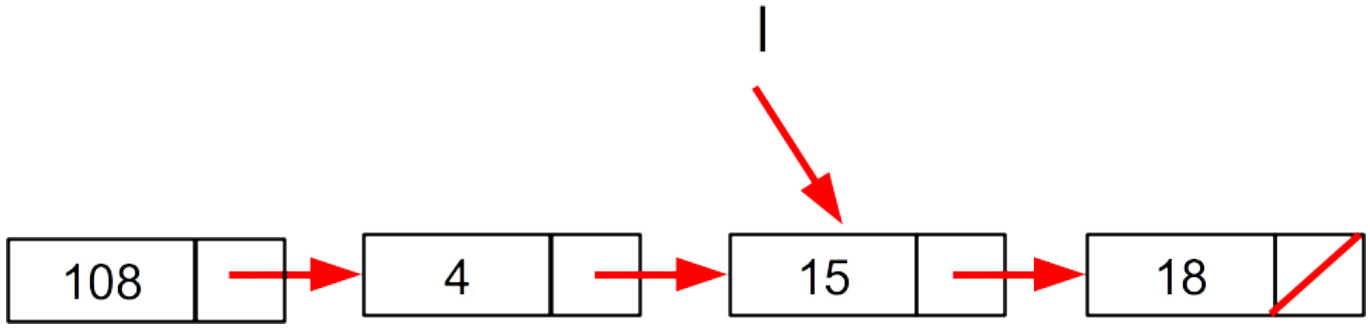


Lista Ligada: Percorrer

Porquê não podemos percorrer a lista utilizando o ponteiro l?

Lista Ligada: Percorrer

Porquê não podemos percorrer a lista utilizando o ponteiro l?

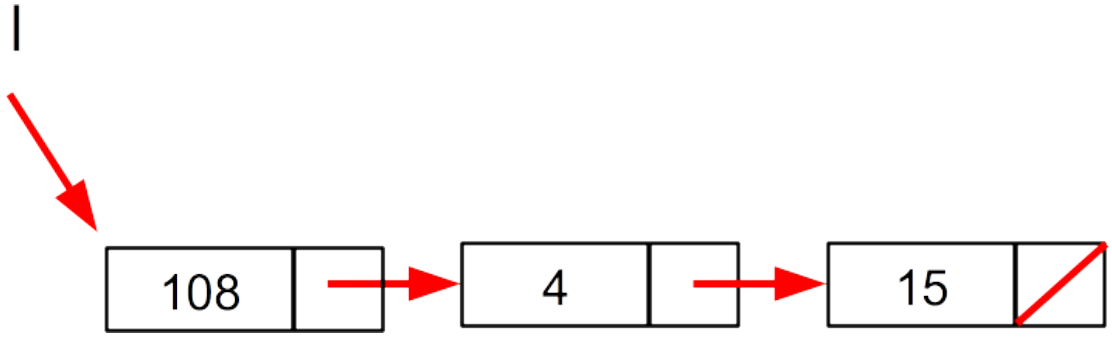


Lista Ligada: Inserir no início

- Criar o novo nó
- Fazer o novo nó apontar para o primeiro
- Atualizar o ponteiro principal da lista

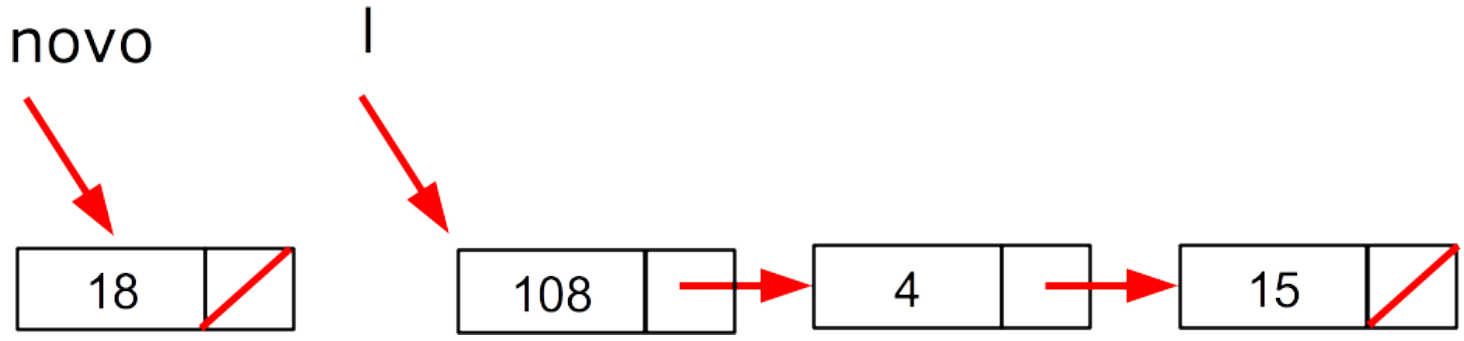
Lista Ligada: Inserir no início

- Criar o novo nó
- Fazer o novo nó apontar para o primeiro
- Atualizar o ponteiro principal da lista



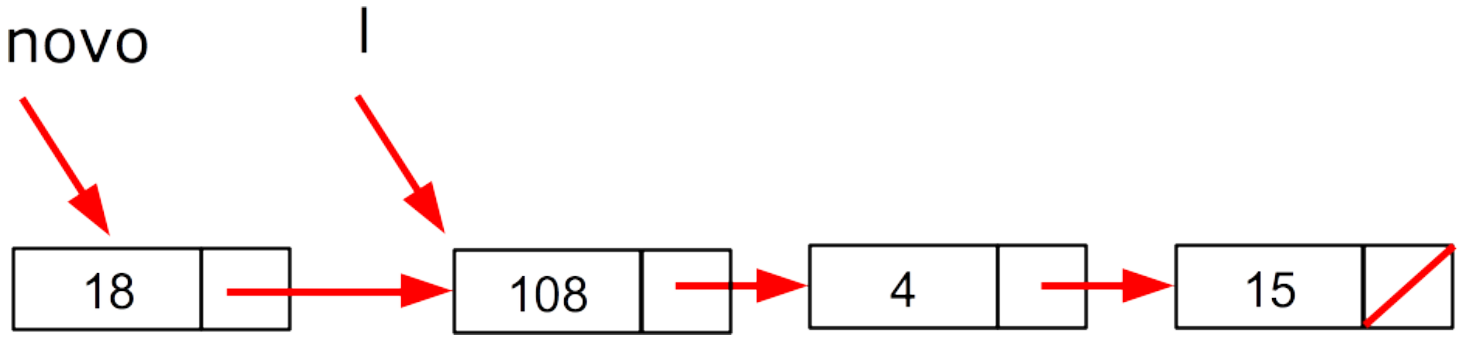
Lista Ligada: Inserir no início

- Criar o novo nó
- Fazer o novo nó apontar para o primeiro
- Atualizar o ponteiro principal da lista



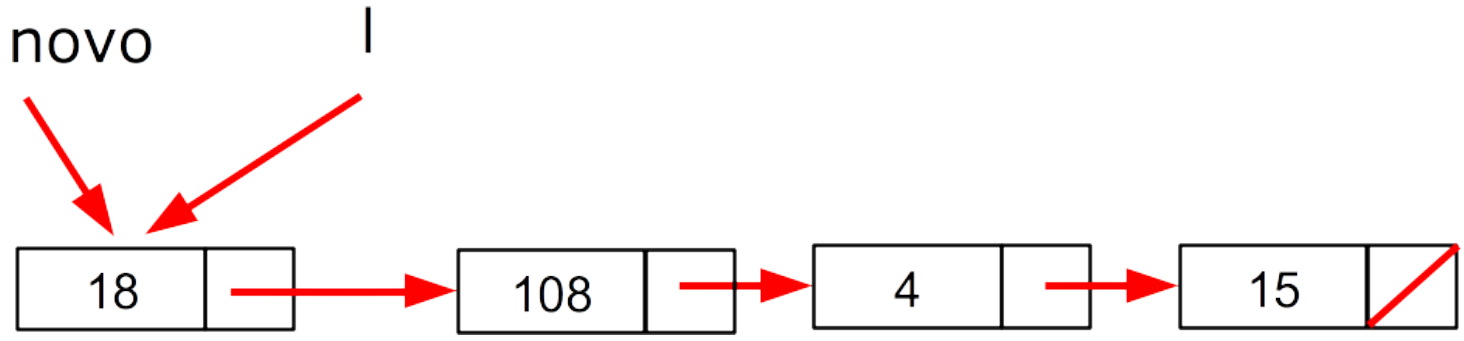
Lista Ligada: Inserir no início

- Criar o novo nó
- Fazer o novo nó apontar para o primeiro
- Atualizar o ponteiro principal da lista



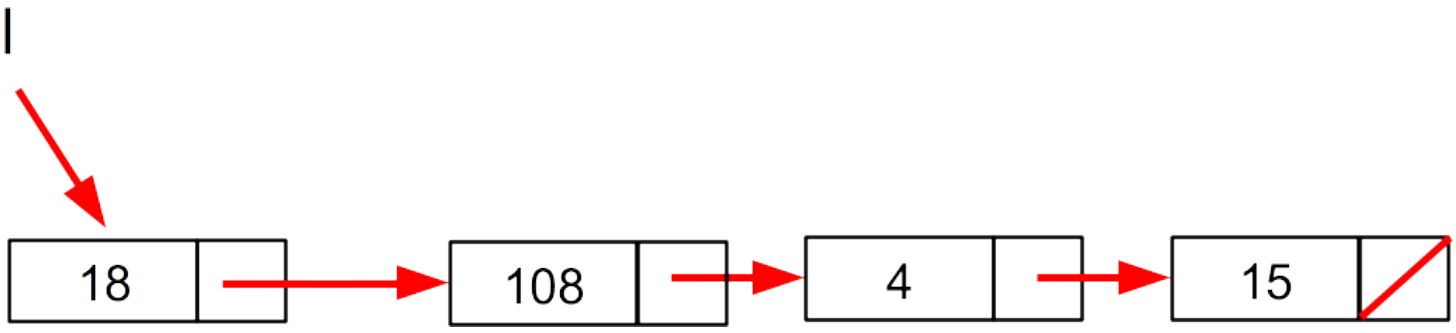
Lista Ligada: Inserir no início

- Criar o novo nó
- Fazer o novo nó apontar para o primeiro
- Atualizar o ponteiro principal da lista



Lista Ligada: Inserir no início

- Criar o novo nó
- Fazer o novo nó apontar para o primeiro
- Atualizar o ponteiro principal da lista

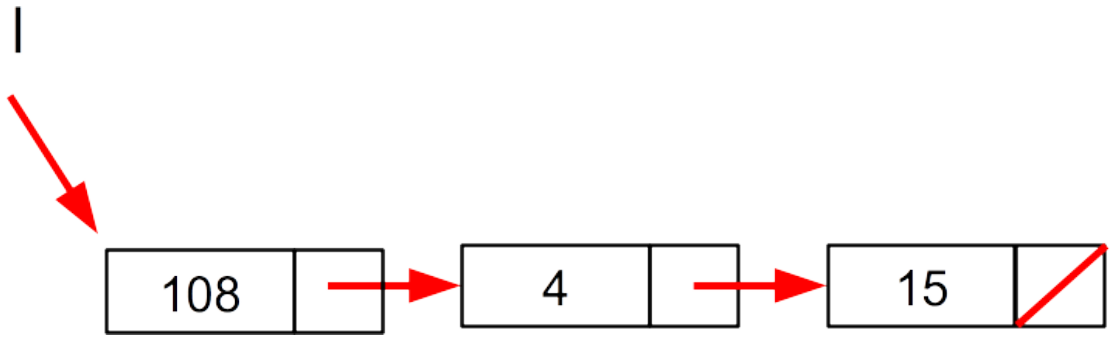


Lista Ligada: Inserir no início

```
void InsereInicio(No **l, int n) {
    // cria novo no
    No*novo;
    novo =(No *) malloc(sizeof(No));
    novo->dado = n;
    novo->prox = NULL;

    // fazer novo no apontar para o primeiro
    novo->prox = *l;

    // atualizar o ponteiro principal
    *l = novo;

    return;
}
```

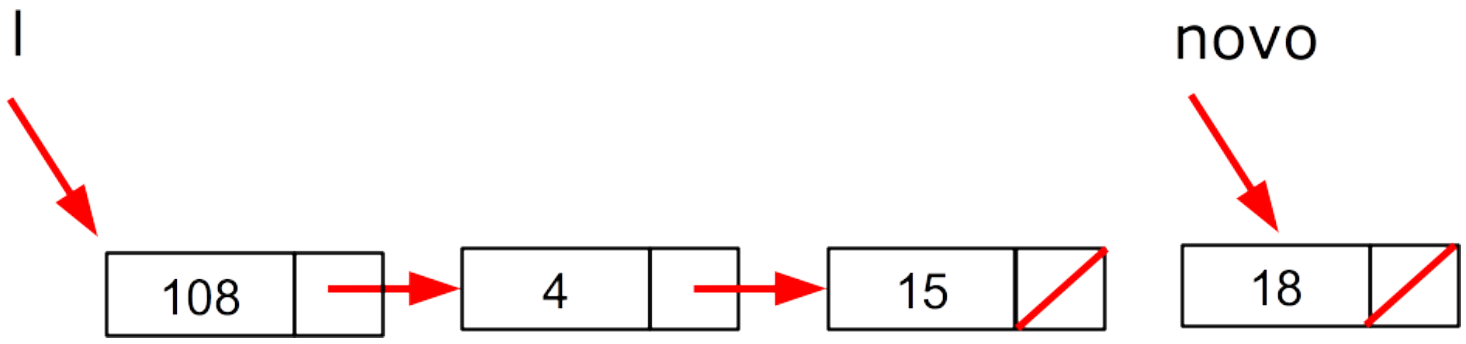
Lista Ligada: Inserir no fim

- Criar o novo nó
- Procurar último nó (o que aponta para NULL)
- Fazer o último nó apontar para o novo



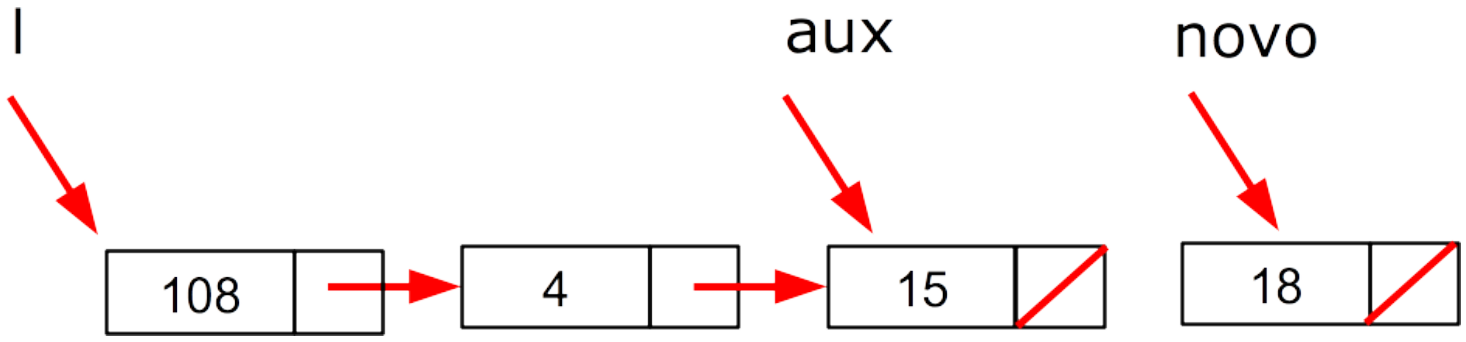
Lista Ligada: Inserir no fim

- Criar o novo nó
- Procurar último nó (o que aponta para NULL)
- Fazer o último nó apontar para o novo



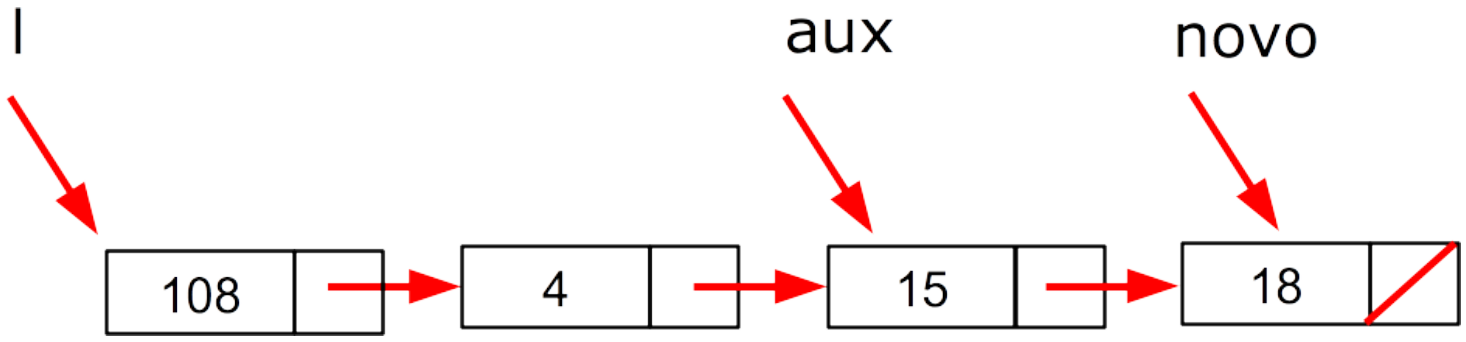
Lista Ligada: Inserir no fim

- Criar o novo nó
- Procurar último nó (o que aponta para NULL)
- Fazer o último nó apontar para o novo



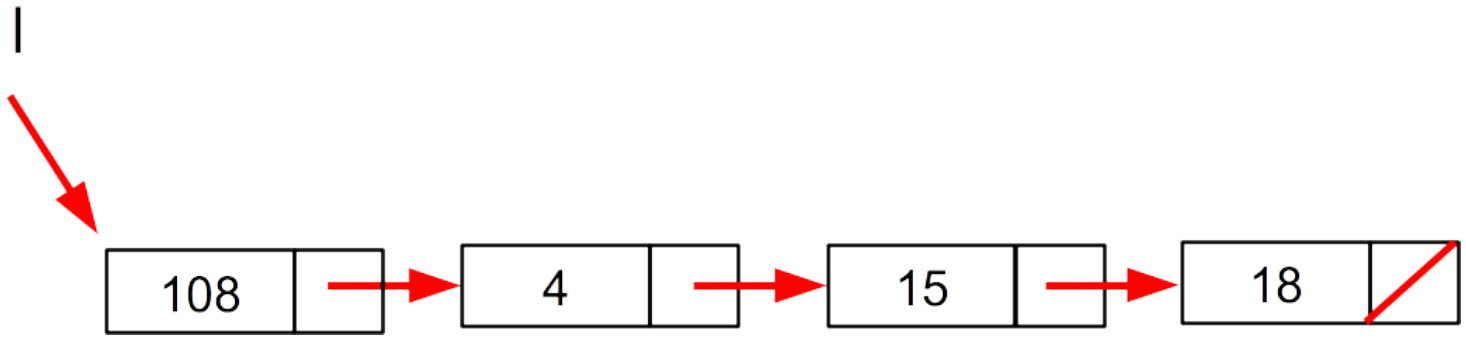
Lista Ligada: Inserir no fim

- Criar o novo nó
- Procurar último nó (o que aponta para NULL)
- Fazer o último nó apontar para o novo



Lista Ligada: Inserir no fim

- Criar o novo nó
- Procurar último nó (o que aponta para NULL)
- Fazer o último nó apontar para o novo



Lista Ligada: Inserir no fim

```
void InsereFim(No **l, int n) {
    // cria novo no
    No*novo;
    novo = (No *) malloc(sizeof(No));
    novo->dado = n;
    novo->prox = NULL;

    // procura ultimo no (aponta pra null)
    No*aux = *l;
    while(aux->prox != NULL) aux = aux->prox;

    // ultimo apontar para o novo no
    aux->prox = novo;

    return;
}
```

Lista Ligada: Inserir no fim

```
void InsereFim(No **l, int n) {
    // cria novo no
    No*novo;
    novo = (No *) malloc(sizeof(No));
    novo->dado = n;
    novo->prox = NULL;

    // procura ultimo no (aponta pra null)
    No*aux = *l;
    while(aux->prox != NULL) aux = aux->prox;

    // ultimo apontar para o novo no
    aux->prox = novo;

    return;
}
```

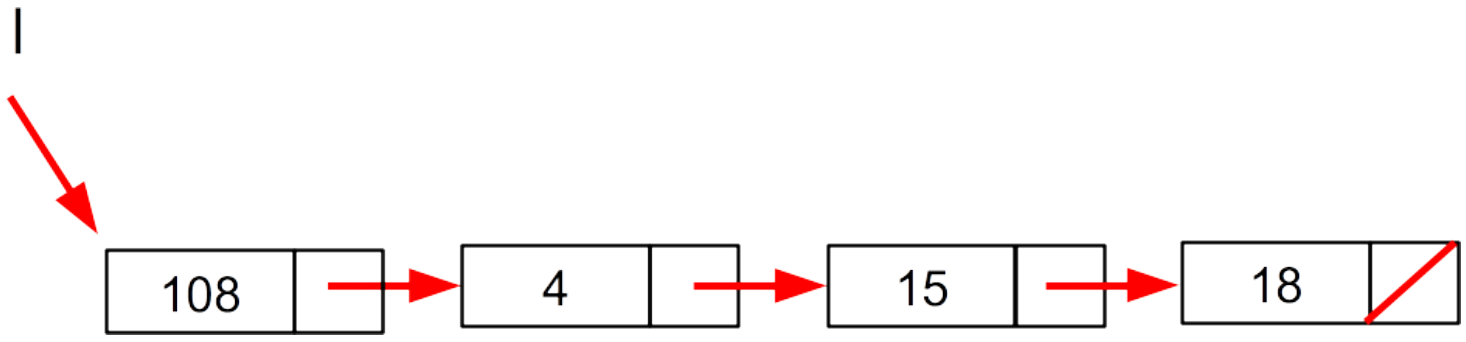
E se a lista estiver vazia?

Lista Ligada: Inserir no fim

```
void InsereFim(No **l, int n) {
    // cria novo no
    No*novo;
    novo = (No *) malloc(sizeof(No));
    novo->dado = n;
    novo->prox = NULL;

    // lista vazia
    if (*l == NULL)(*l) = novo;

    else {
        // procura ultimo no (aponta pra null)
        No*aux = *l;
        while(aux->prox != NULL)aux = aux->prox;

        // ultimo apontar para o novo no
        aux->prox = novo;
    }

    return;
}
```

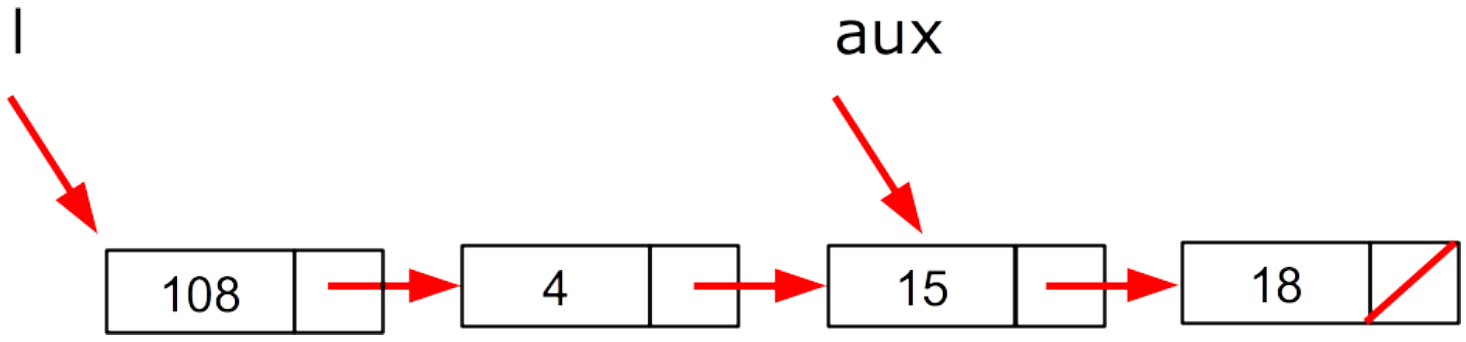
Lista Ligada: Remover no fim

- Procurar penúltimo nó (o que aponta para o que aponta para NULL)
- Liberar a memória do último nó (o nó apontado pelo penúltimo)
- Fazer o penúltimo nó (novo último) apontar para NULL



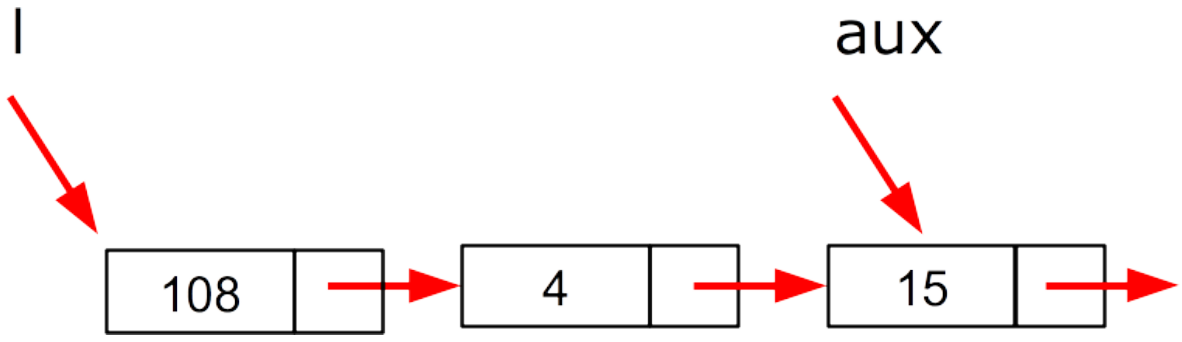
Lista Ligada: Remover no fim

- Procurar penúltimo nó (o que aponta para o que aponta para NULL)
- Liberar a memória do último nó (o nó apontado pelo penúltimo)
- Fazer o penúltimo nó (novo último) apontar para NULL



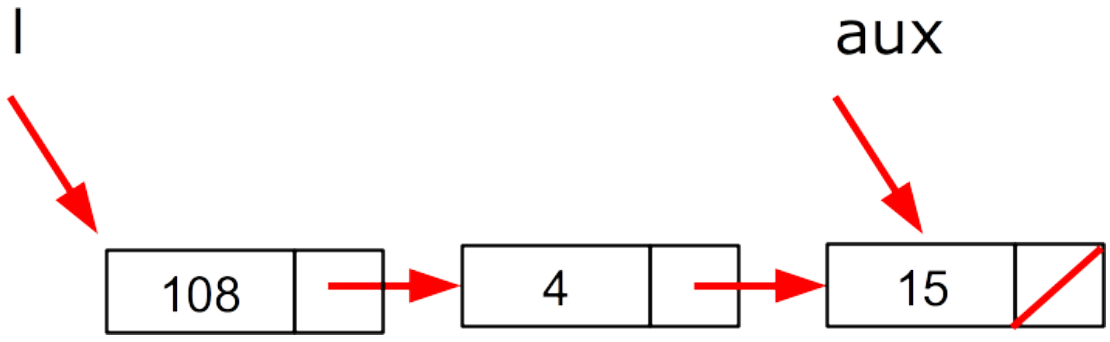
Lista Ligada: Remover no fim

- Procurar penúltimo nó (o que aponta para o que aponta para NULL)
- Liberar a memória do último nó (o nó apontado pelo penúltimo)
- Fazer o penúltimo nó (novo último) apontar para NULL



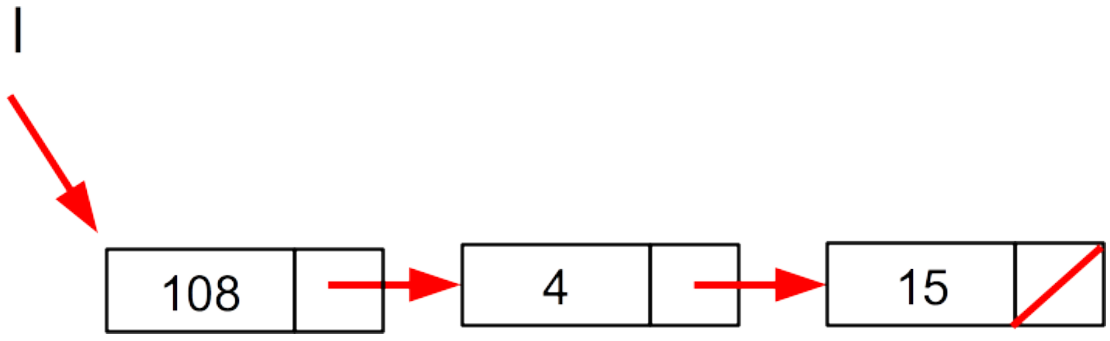
Lista Ligada: Remover no fim

- Procurar penúltimo nó (o que aponta para o que aponta para NULL)
- Liberar a memória do último nó (o nó apontado pelo penúltimo)
- Fazer o penúltimo nó (novo último) apontar para NULL



Lista Ligada: Remover no fim

- Procurar penúltimo nó (o que aponta para o que aponta para NULL)
- Liberar a memória do último nó (o nó apontado pelo penúltimo)
- Fazer o penúltimo nó (novo último) apontar para NULL



Lista Ligada: Remover no fim

```
void RemoveFim(No **l, int n) {
    No* aux;
    if (*l == NULL) return;

    // procura penultimo no (aponta pra o ultimo)
    aux = *l;
    while(aux->prox->prox != NULL) aux = aux->prox;

    // libera memoria do ultimo no
    free(aux->prox);

    // faz o novo ultimo apontar para NULL
    aux->prox = NULL;

    return;
}
```

Lista Ligada: Remover no fim

```
void RemoveFim(No **l, int n) {
    No* aux;
    if (*l == NULL) return;

    // procura penultimo no (aponta pra o ultimo)
    aux = *l;
    while(aux->prox->prox != NULL) aux = aux->prox;

    // libera memoria do ultimo no
    free(aux->prox);

    // faz o novo ultimo apontar para NULL
    aux->prox = NULL;

    return;
}
```

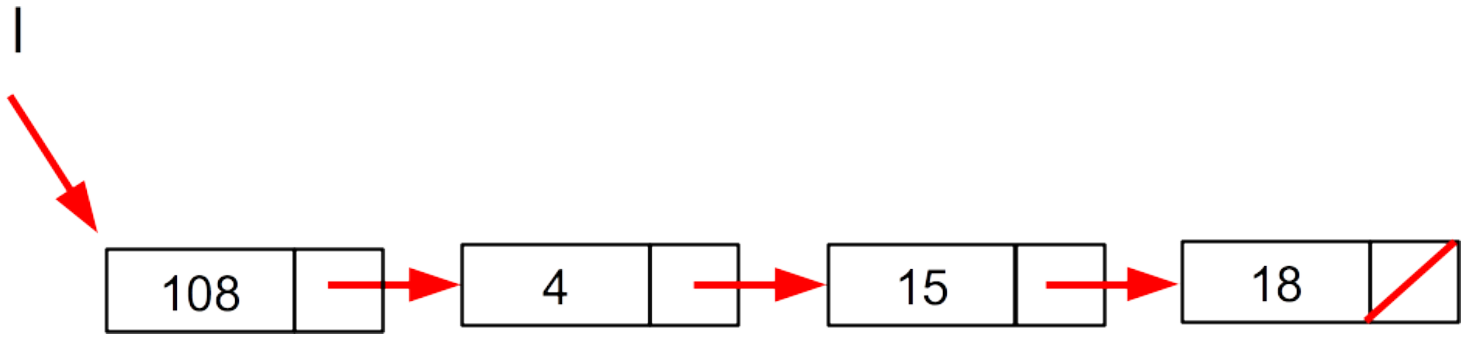
E se a lista tiver somente um elemento?

Lista Ligada: Remover no fim

```
void RemoveFim(No**l, int n){
    No* aux;
    if (*l == NULL) return;

    // checa se tem somente umelemento if
    (*l->prox == NULL){
        free(*l);
        *l = NULL;
    }

    else {
        // procura penultimo no (aponta pra o ultimo)
        aux = *l;
        while(aux->prox->prox != NULL) aux = aux->prox;
        // libera memoria do ultimo no
        free(aux->prox);
        // faz o novo ultimo apontar para NULL
        aux->prox = NULL;
    }
    return;
}
```

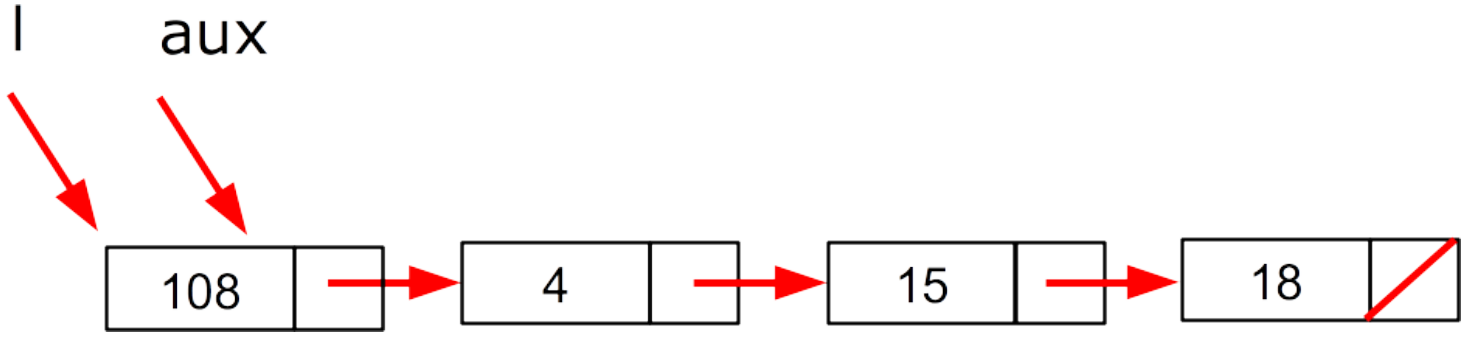
Lista Ligada: Remover no início

- Copiar o ponteiro principal da lista
- Fazer o ponteiro principal da lista apontar pro próximo (novo primeiro elemento)
- Liberar a memória usando a cópia feita no ponteiro auxiliar



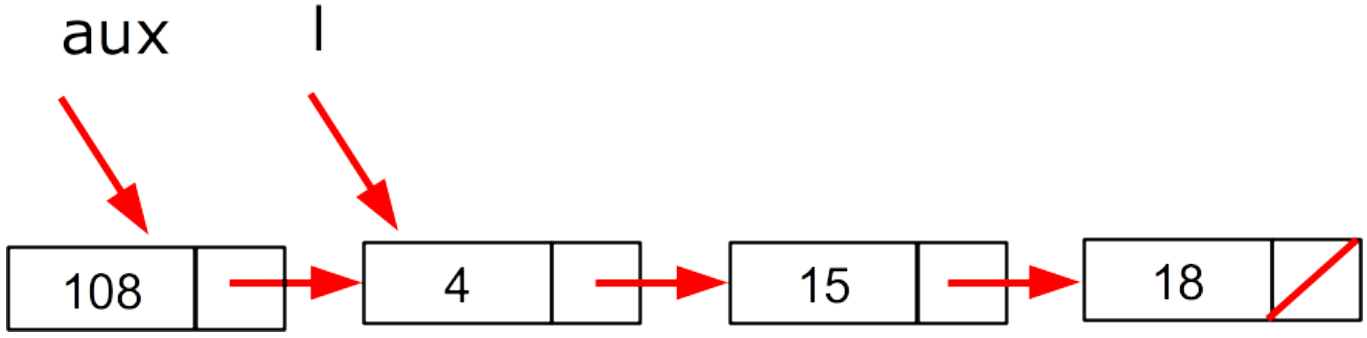
Lista Ligada: Remover no início

- Copiar o ponteiro principal da lista
- Fazer o ponteiro principal da lista apontar pro próximo (novo primeiro elemento)
- Liberar a memória usando a cópia feita no ponteiro auxiliar



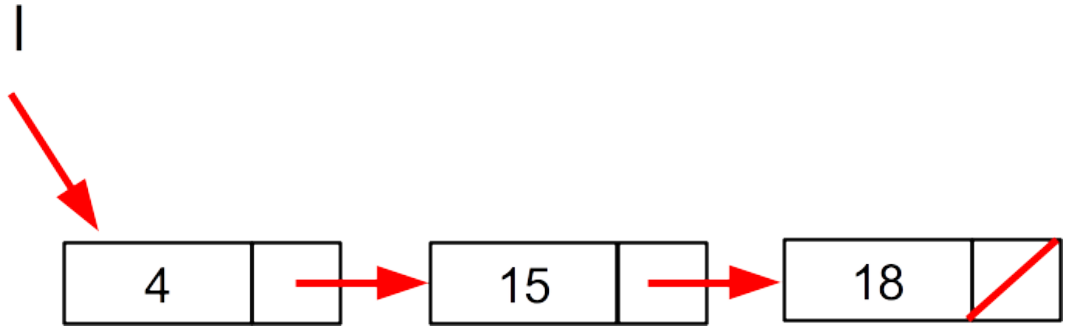
Lista Ligada: Remover no início

- Copiar o ponteiro principal da lista
- Fazer o ponteiro principal da lista apontar pro próximo (novo primeiro elemento)
- Liberar a memória usando a cópia feita no ponteiro auxiliar



Lista Ligada: Remover no início

- Copiar o ponteiro principal da lista
- Fazer o ponteiro principal da lista apontar pro próximo (novo primeiro elemento)
- Liberar a memória usando a cópia feita no ponteiro auxiliar



Lista Ligada: Remover no início

```
void RemoveInicio(No **l, int n) {
    No* aux;
    if (*l == NULL) return;

    // copiar o ponteiro principal
    aux = *l;

    // fazer o ponteiro principal apontar pro proximo
    *l = *l->prox;

    // liberar a memoria
    free(aux);

    return;
}
```

Lista Ligada: TAD

- Criar uma lista vazia
- Inserir um elemento no fim ou início
- Remover um elemento no início ou fim
- Checar se está vazia
- Imprimir
- Calcular tamanho (número de elementos)

Lista Ligada vs Vetor

Lista Ligada:

- É esparça na memória
- Faz a alocação para cada elemento
- É difícil acessar o i-ésimo elemento
- Inserir e remover elementos no meio é simples
- Precisa de memória extra para os ponteiros

Vetor Dinâmico:

- É sempre contínuo na memória
- Aloca todos os elementos de uma vez
- Acesso fácil ao i-ésimo elemento
- Inserir e remover elementos no meio é trabalhoso e caro
- Ocupa o espaço exato na memória

Exercício

Faça uma função que checa se uma lista está vazia e outra que conte o número de elementos de uma lista.

Exercício: Solução

```
int listaVazia(No* l) {
    if (l ==NULL)return 1;
    else return 0;
}
```

```
int tamanhoLista (No* l) {
    int n=0;
    No*aux = l;
    while (aux != NULL){
        n++;
        aux =aux->prox;
    }
    return n;
}
```