Aula 30 – Listas Ligadas

Norton T. Roman & Luciano A. Digiampietri

Listas Ligadas

- Considere que desejamos gerenciar um conjunto de registros de pessoas

Listas Ligadas

- Considere que desejamos gerenciar um conjunto de registros de pessoas

- Criamos uma estrutura chamada `Registro` para armazenar as informações de cada pessoa

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;

typedef struct {
  int id;
  float media;
} Registro;
```

Listas Ligadas

- Considere que desejamos gerenciar um conjunto de registros de pessoas
- Criamos uma função para criar um novo registro
- Criamos uma estrutura chamada `Registro` para armazenar as informações de cada pessoa

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;

typedef struct {
  int id;
  float media;
} Registro;
```

```
Registro novo(int id, float media){
    Registro res;
    res.id = id;
    res.media = media;
    return res;
}
```

Listas Ligadas

- Considere que desejamos gerenciar um conjunto de registros de pessoas
- Criamos uma função para criar um novo registro
- Criamos uma estrutura chamada `Registro` para armazenar as informações de cada pessoa
- E armazenamos os registros em um arranjo

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;

typedef struct {
  int id;
  float media;
} Registro;
```

```
Registro novo(int id, float media){
    Registro res;
    res.id = id;
    res.media = media;
    return res;
}
int main() {
  Registro* cadastro = (Registro*)
              malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);
  return 0;
}
```

Listas Ligadas

- E se, durante a execução, precisássemos cadastrar mais uma pessoa (além dos limites do arranjo)?

```
int main() {
  Registro* cadastro = (Registro*)
              malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);




  return 0;
}
```

Listas Ligadas

- E se, durante a execução, precisássemos cadastrar mais uma pessoa (além dos limites do arranjo)?

```
int main() {
  Registro* cadastro = (Registro*)
              malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);


Precisamos inserir uma terceira pessoa!?!


  return 0;
}
```

Listas Ligadas

- E se, durante a execução, precisássemos cadastrar mais uma pessoa (além dos limites do arranjo)?
  - Não haverá espaço alocado para ela no arranjo

```
int main() {
  Registro* cadastro = (Registro*)
              malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);


Precisamos inserir uma terceira pessoa!?!


  return 0;
}
```

Listas Ligadas

- Que fazer?

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior

```
int main() {

  Registro* cadastro = (Registro*)
              malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];
  maior[2] = novo(9, 7.5);









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa
  - Liberar a memória do arranjo antigo

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];
  maior[2] = novo(9, 7.5);









  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa
  - Liberar a memória do arranjo antigo

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];
  maior[2] = novo(9, 7.5);

  free(cadastro);



  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa
  - Liberar a memória do arranjo antigo
  - Substituir o arranjo antigo pelo maior

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];
  maior[2] = novo(9, 7.5);

  free(cadastro);



  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa
  - Liberar a memória do arranjo antigo
  - Substituir o arranjo antigo pelo maior

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];
  maior[2] = novo(9, 7.5);

  free(cadastro);

  cadastro = maior;

  return 0;
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova pessoa
  - Liberar a memória do arranjo antigo
  - Substituir o arranjo antigo pelo maior
- Incrivelmente ineficiente

```
int main() {

  Registro* cadastro = (Registro*)
             malloc(sizeof(Registro)*2);
  cadastro[0] = novo(4, 8.5);
  cadastro[1] = novo(7, 6.5);

  Registro* maior = (Registro*)
             malloc(sizeof(Registro)*3);
  maior[0] = cadastro[0];
  maior[1] = cadastro[1];
  maior[2] = novo(9, 7.5);

  free(cadastro);

  cadastro = maior;

  return 0;
}
```

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não
- Alternativa: Lista Ligada

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não
- Alternativa: Lista Ligada

> Uma lista ligada é uma lista onde cada elemento – chamado de nó – contém um valor e uma referência para o elemento seguinte na lista

Listas Ligadas

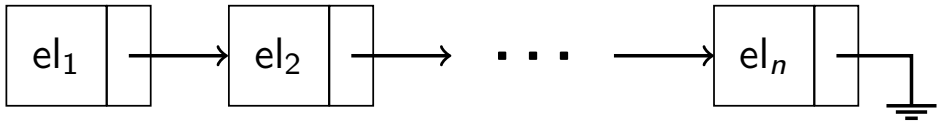
- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não
- Alternativa: Lista Ligada

> Uma lista ligada é uma lista onde cada elemento – chamado de nó – contém um valor e uma referência para o elemento seguinte na lista

  - Assim, sabendo onde está o primeiro elemento da lista, podemos chegar a qualquer outro elemento

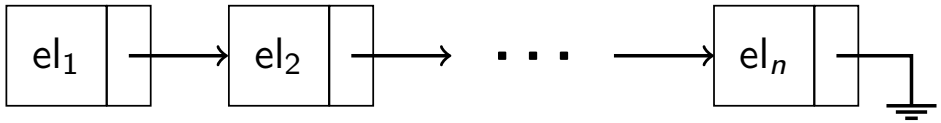
Listas Ligadas

- Simples:

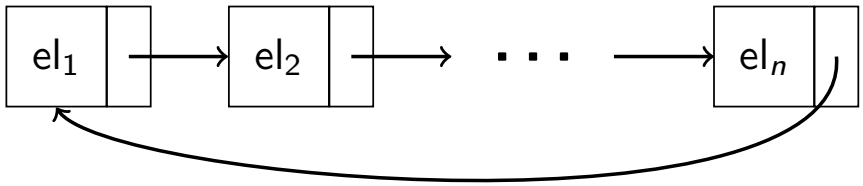


Listas Ligadas

- Simples:

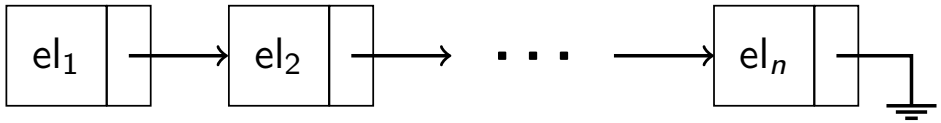


- Circular:

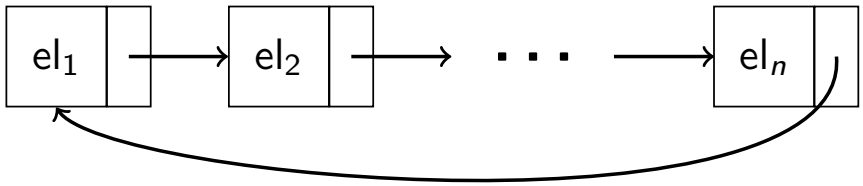


Listas Ligadas

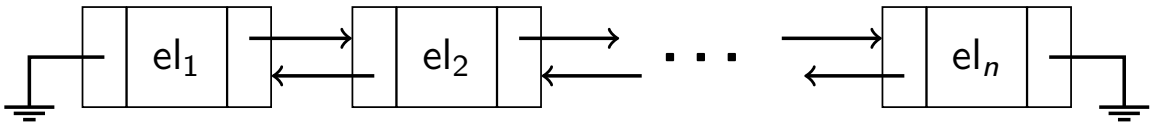
- Simples:



- Circular:

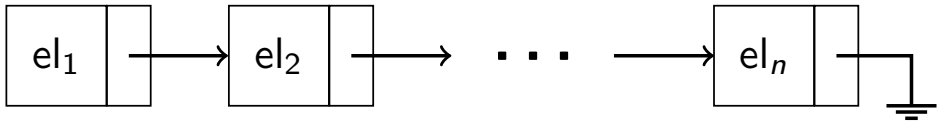


- Duplamente ligada:

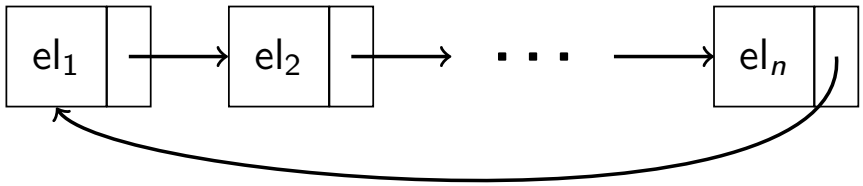


Listas Ligadas

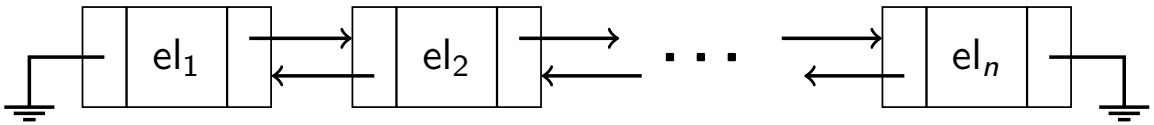
- Simples:



- Circular:



- Duplamente ligada:



- Etc

Listas Ligadas

- Dado armazenado:

Listas Ligadas

- Dado armazenado:
  - Dados da pessoa
    `Registro`

Listas Ligadas

- Dado armazenado:
  - Dados da pessoa
    `Registro`
- Conteúdo para o algoritmo:

```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;
```

Listas Ligadas

- Dado armazenado:
  - Dados da pessoa
    `Registro`
- Conteúdo para o algoritmo:
  - Temos o dado armazenado

```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;
```

Listas Ligadas

- Dado armazenado:
  - Dados da pessoa
    `Registro`
- Conteúdo para o algoritmo:
  - Temos o dado armazenado
  - E uma referência para o próximo elemento

```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;
```

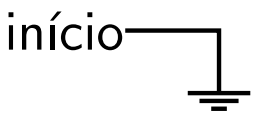
Listas Ligadas

- Criando uma lista simples:

```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

- Criando uma lista simples:

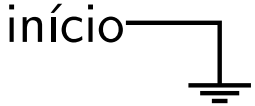



```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

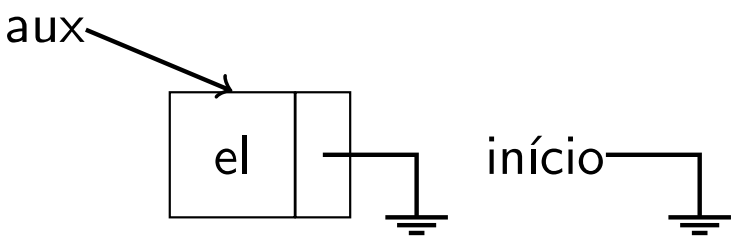
- Inserindo elemento no início:

```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```



Listas Ligadas

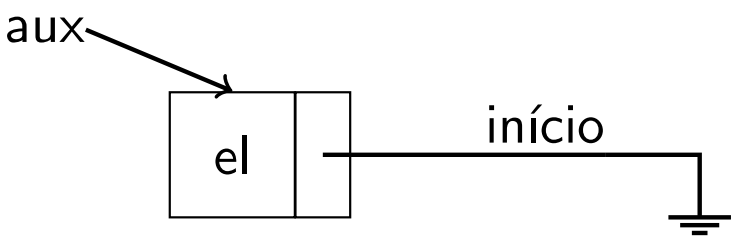
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

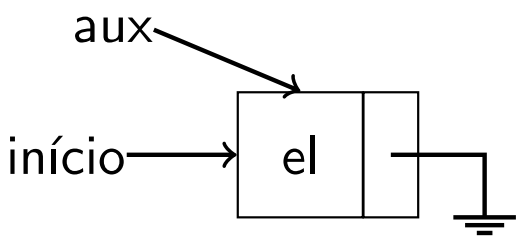
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido
  - Faça o elemento do início da lista ser seu próximo




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

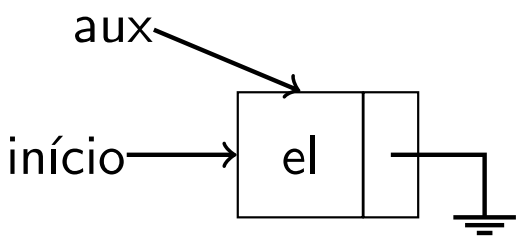
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido
  - Faça o elemento do início da lista ser seu próximo
  - Torne esse novo elemento o novo início da lista




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

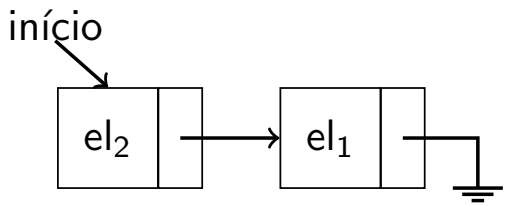
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido
  - Faça o elemento do início da lista ser seu próximo
  - Torne esse novo elemento o novo início da lista
  - Incremente o número de elementos




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

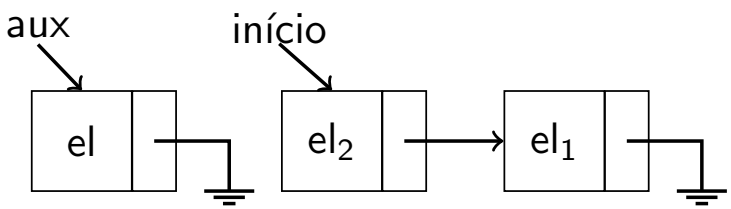
- E se a lista já tiver algo?




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

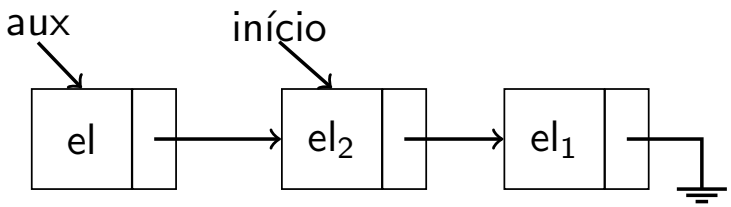
- E se a lista já tiver algo?




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

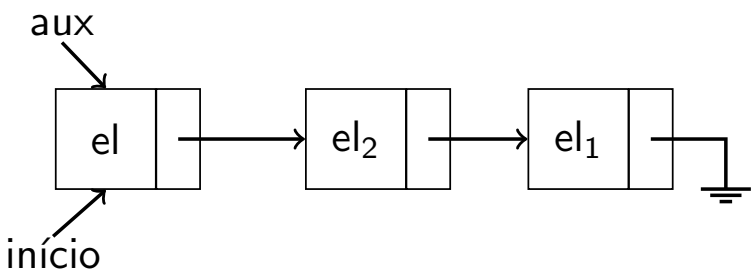
- E se a lista já tiver algo?




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

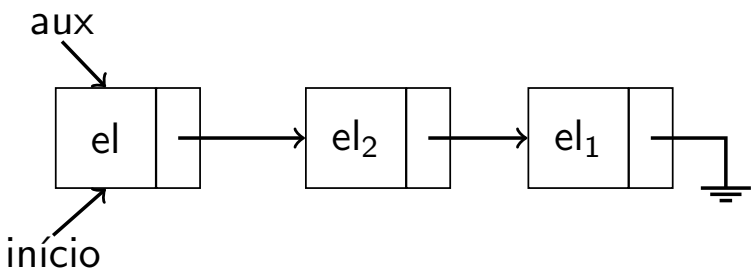
- E se a lista já tiver algo?




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

- E se a lista já tiver algo?




```
typedef struct aux{
  Registro reg;
  struct aux* prox;
} No;

typedef struct {
  No* inicio;
  int elementos;
} Lista;

void inicializa(Lista* l) {
  l->inicio = NULL;
  l->elementos = 0;
}

void insere(Lista* l, Registro reg) {
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  novo->prox = l->inicio;
  l->inicio = novo;
  l->elementos++;
}
```

Listas Ligadas

- E como podemos usar isso?

Listas Ligadas

- E como podemos usar isso?

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!
  - Não precisamos de variável auxiliar para guardar o resultado de uma função

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!
  - Não precisamos de variável auxiliar para guardar o resultado de uma função
  - O computador irá buscar o registro `reg`, de `n`

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!
  - Não precisamos de variável auxiliar para guardar o resultado de uma função
  - O computador irá buscar o registro `reg`, de `n`
  - Então olhará o campo `id`, de `reg`

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

Saída

```
9
7
4
```

```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Listas Ligadas

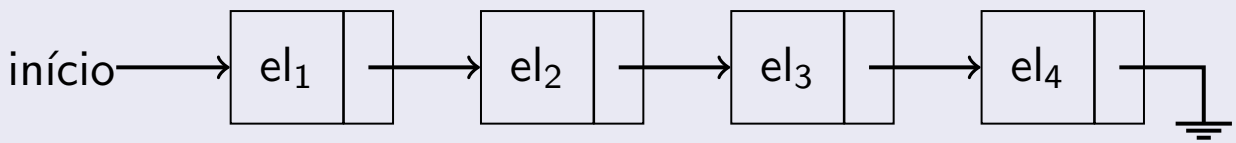
Saída

```
9
7
4
```

(Note que a ordem está inversa à de inserção)

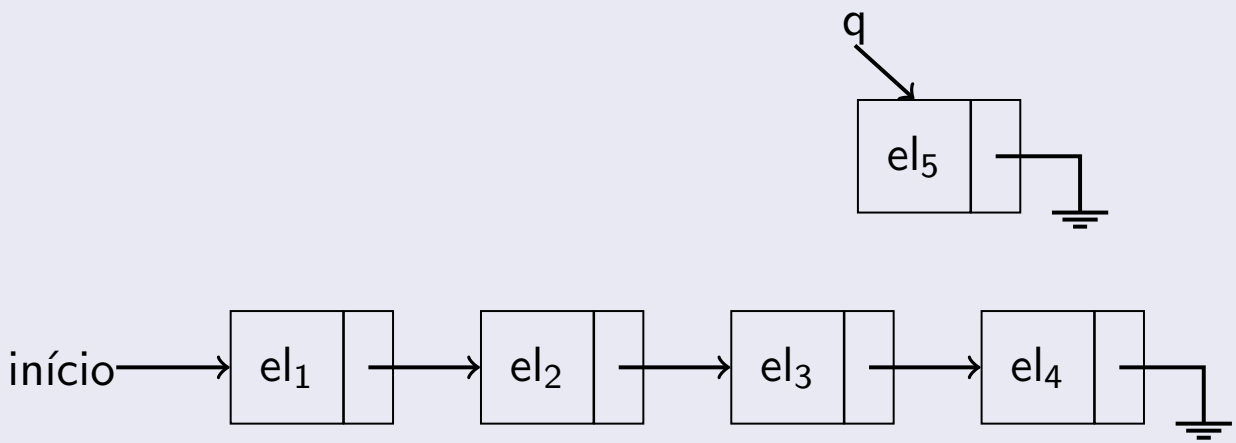
```
int main() {
  Lista l1;
  inicializa(&l1);

  insere(&l1,novo(4, 8.5));
  insere(&l1,novo(7, 6.5));
  insere(&l1,novo(9, 7.5));

  No* n = l1.inicio;
  while (n != NULL) {
    printf("%i\n", n->reg.id);
    n = n->prox;
  }

  return 0;
}
```

Incluindo Elemento em uma Posição (4^a)



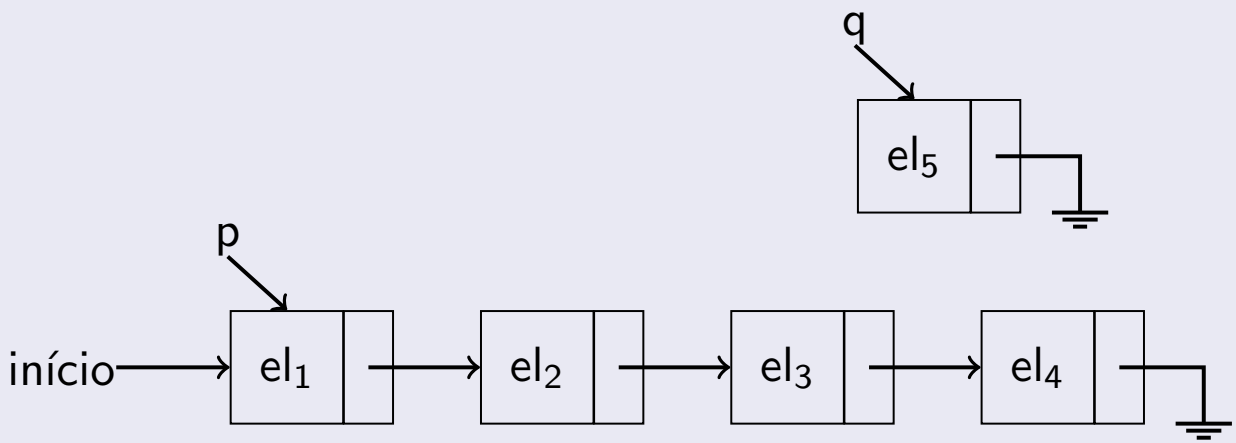
Incluindo Elemento em uma Posição (4^a)

- Alocamos espaço para o novo elemento



Incluindo Elemento em uma Posição (4^a)

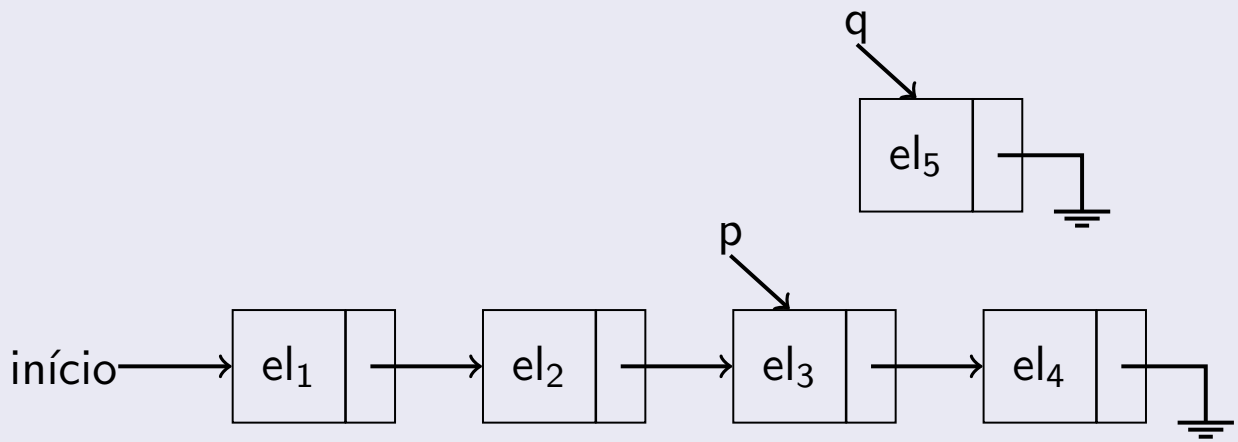
- Alocamos espaço para o novo elemento
- Marcamos o início da lista



Listas Ligadas

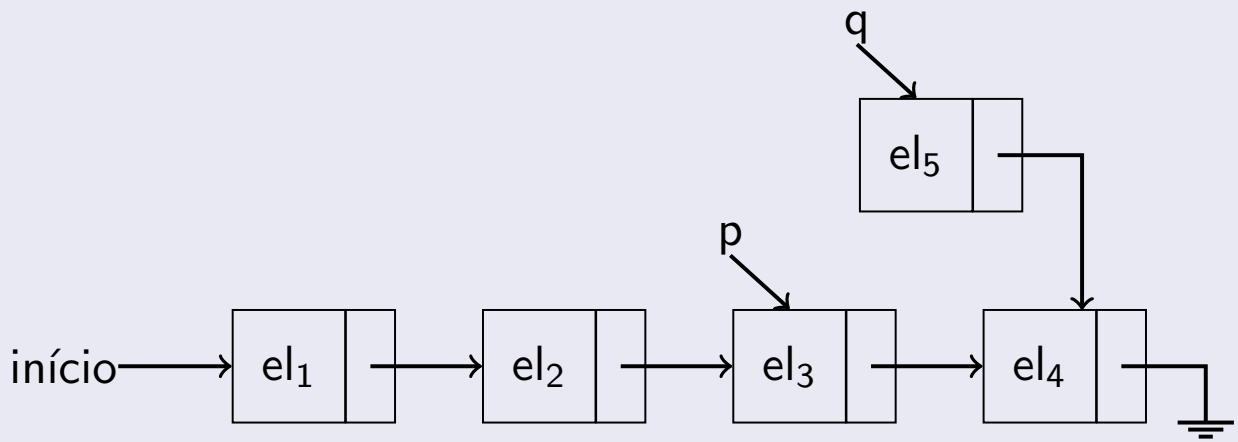
Incluindo Elemento em uma Posição (4^a)

- Alocamos espaço para o novo elemento
- Marcamos o início da lista
- Andamos até a (n-1)^a posição:



Incluindo Elemento em uma Posição (4^a)

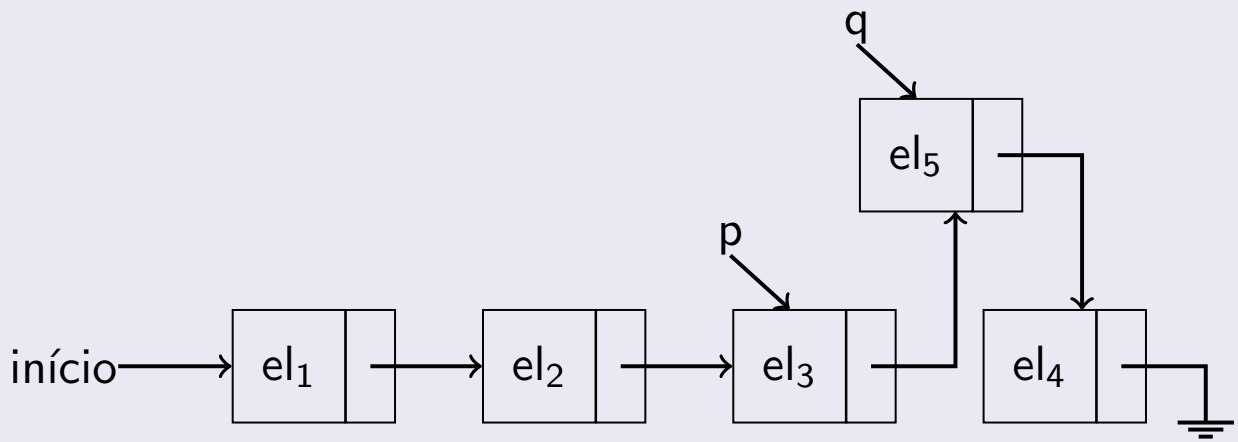
- Fazemos o campo prox do novo elemento apontar para o nº elemento da lista



Listas Ligadas

Incluindo Elemento em uma Posição (4^a)

- Fazemos o campo prox do novo elemento apontar para o nº elemento da lista
- Fazemos o elemento seguinte ao (n-1)º ser o novo elemento



Listas Ligadas

- Isso, contudo, funciona para posições > 0

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

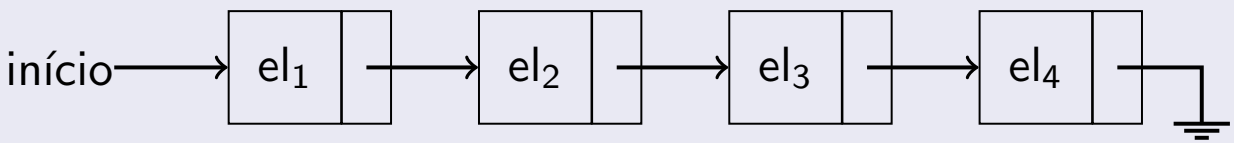
```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Listas Ligadas

- Isso, contudo, funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

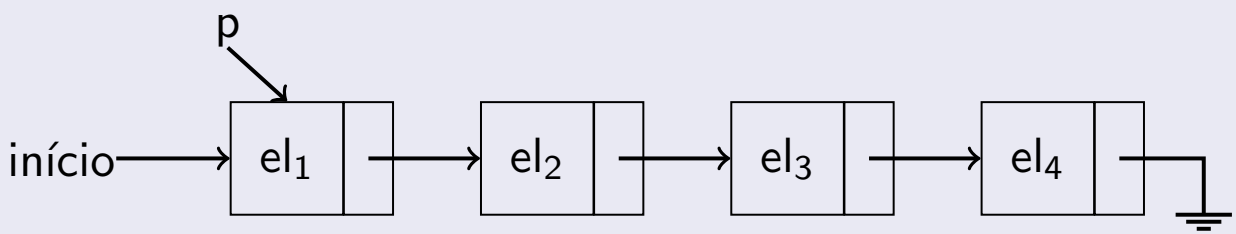
```
bool insere(Lista* l, Registro reg, int pos) {
  if (pos<0 || pos>l->elementos) return false;
  No* novo = (No*) malloc(sizeof(No));
  novo->reg = reg;
  int i;
  No* p;
  if (pos == 0){
    novo->prox = l->inicio;
    l->inicio = novo;
  }else{
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    novo->prox = p->prox;
    p->prox = novo;
  }
  l->elementos++;
  return true;
}
```

Excluindo Elemento em uma Posição (3^a)



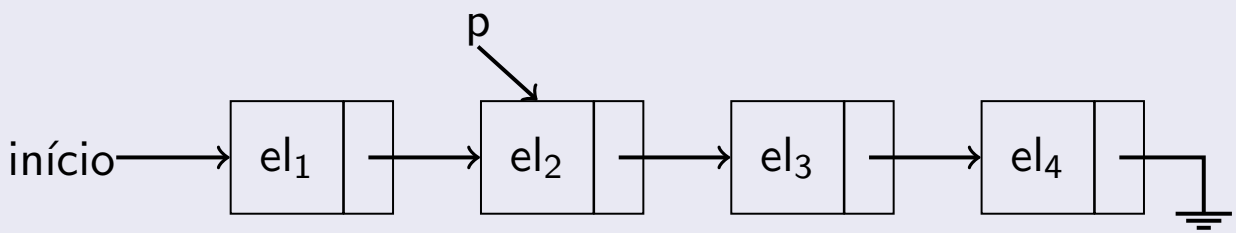
Excluindo Elemento em uma Posição (3^a)

- Marcamos o início da lista



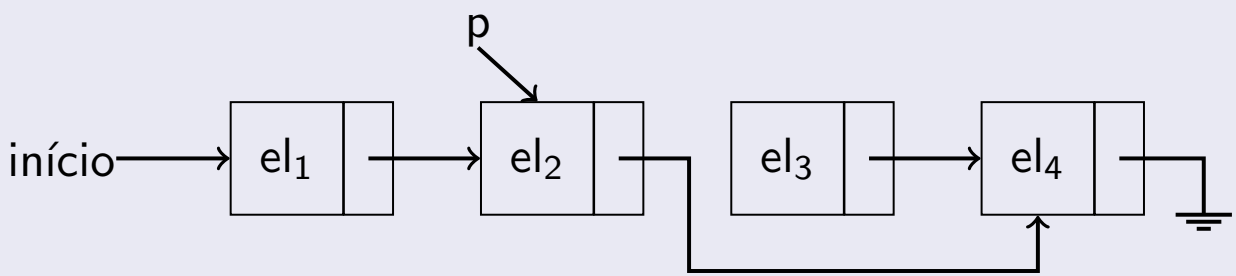
Excluindo Elemento em uma Posição (3ª)

- Marcamos o início da lista
- Movemos até o elemento anterior ao nº



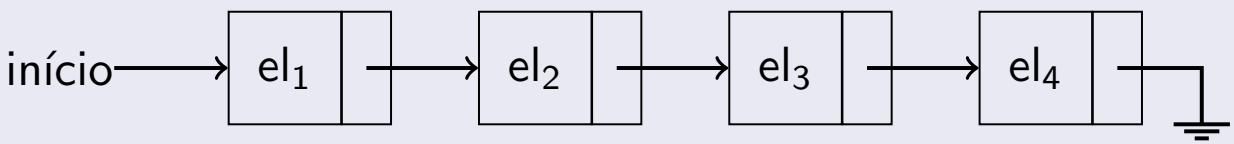
Excluindo Elemento em uma Posição (3^a)

- Marcamos o início da lista
- Movemos até o elemento anterior ao nº
- Fazemos o próximo elemento de p ser o elemento que está após seu próximo



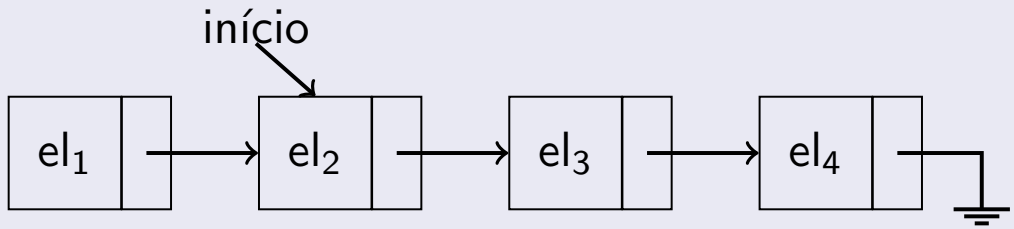
Excluindo Elemento em uma Posição (3^a)

- E se a posição pretendida for a primeira (0)?



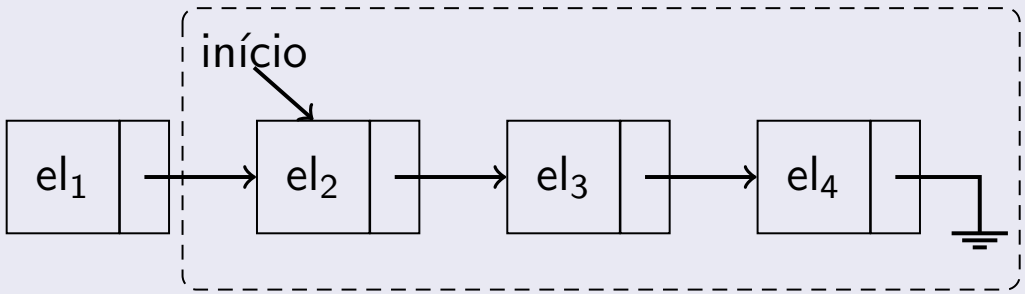
Excluindo Elemento em uma Posição (3^a)

- E se a posição pretendida for a primeira (0)?
- Movemos a início



Excluindo Elemento em uma Posição (3^a)

- E se a posição pretendida for a primeira (0)?
- Movemos a início
- A lista fica



Listas Ligadas

- Código:

```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Listas Ligadas

- Código:

```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Listas Ligadas

- Código:

```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Listas Ligadas

- Código:

```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Listas Ligadas

- Código:

```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Listas Ligadas

- Código:

```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Listas Ligadas

- Código:

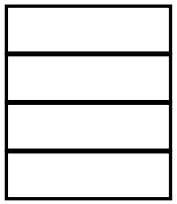
```
bool exclui(Lista* l, int pos) {
  if (pos<0 || pos>l->elementos-1) return false;
  int i;
  No* p;
  No* apagar;
  if (pos == 0) {
    apagar = l->inicio;
    l->inicio = apagar->prox;
  }else {
    p = l->inicio;
    for (i=0;i<pos-1;i++) p = p->prox;
    apagar = p->prox;
    p->prox = apagar->prox;
  }
  free(apagar);
  l->elementos--;
  return true;
}
```

Pilhas

- Estruturas muito usadas em computação

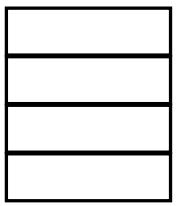
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros



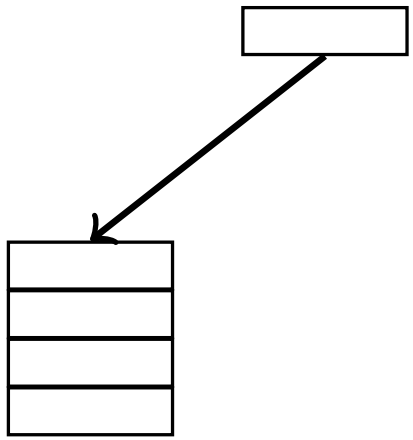
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?



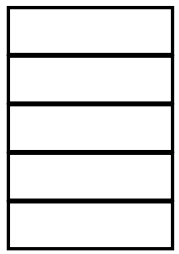
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?



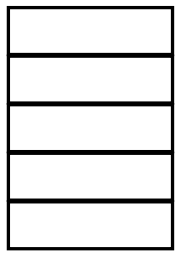
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?



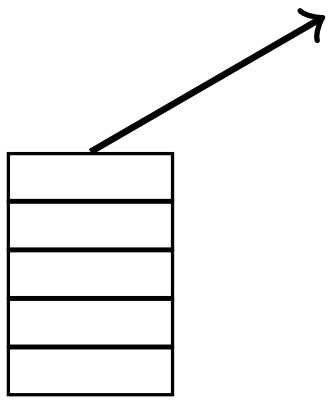
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?
  - Se queremos tirar um livro, de onde tiramos?



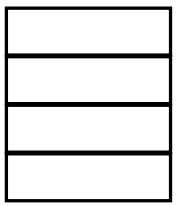
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?
  - Se queremos tirar um livro, de onde tiramos?



Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?
  - Se queremos tirar um livro, de onde tiramos?



Pilhas

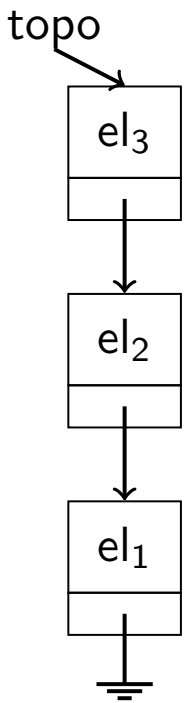
- E como representamos isso?

Pilhas

- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:

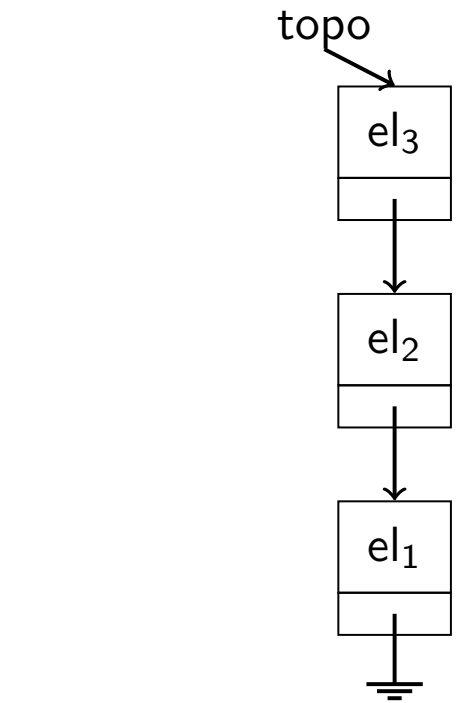
Pilhas

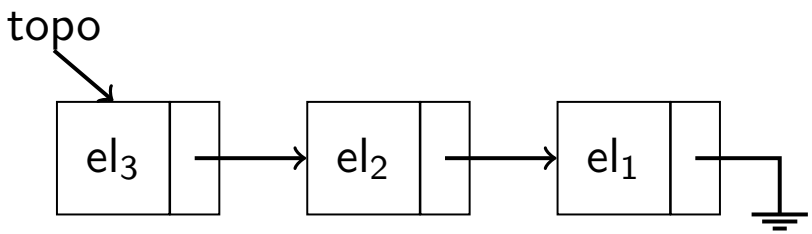
- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:



Pilhas

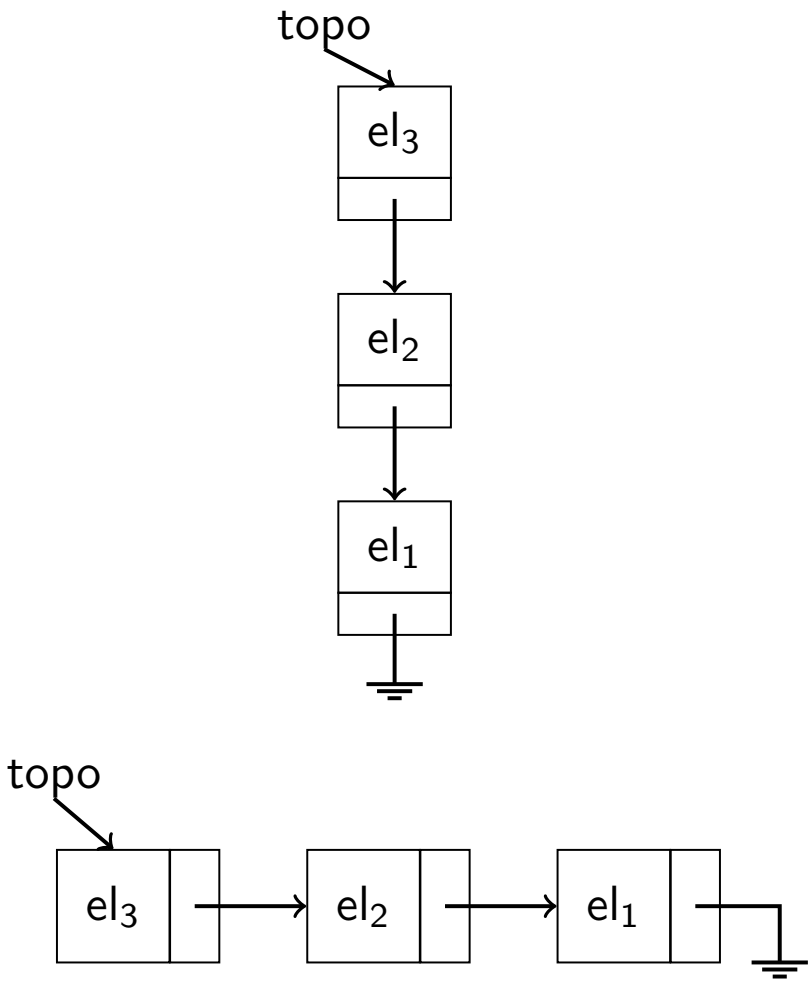
- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:





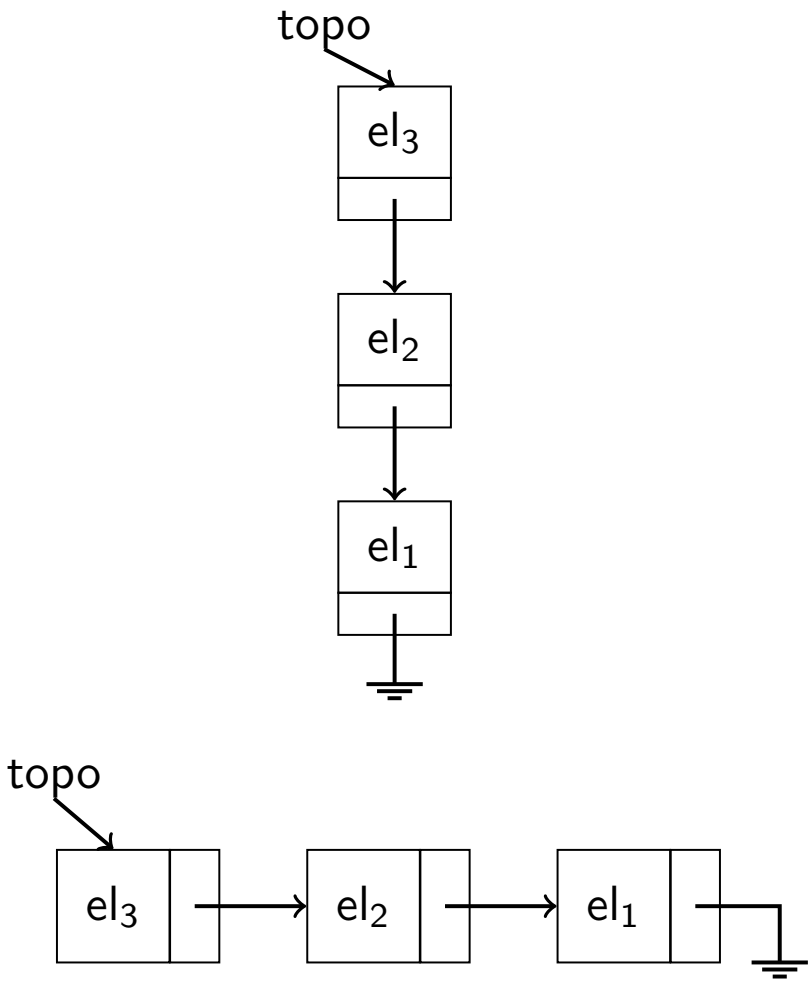
Pilhas

- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:
- Empilhamentos e Desempilhamentos são feitos sempre na posição 0



Pilhas

- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:
- Empilhamentos e Desempilhamentos são feitos sempre na posição 0
  - O topo da pilha



Filas

- Pilhas são do tipo LIFO (Last In First Out)

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair
- Uma estrutura alternativa à pilha é a Fila

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair
- Uma estrutura alternativa à pilha é a Fila
  - FIFO (First In First Out)

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair
- Uma estrutura alternativa à pilha é a Fila
  - FIFO (First In First Out)
  - O primeiro item a entrar é o primeiro a sair

Filas

- Pense numa fila de banco

Filas

- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?

Filas

- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?
  - Quem será o primeiro a ser atendido?

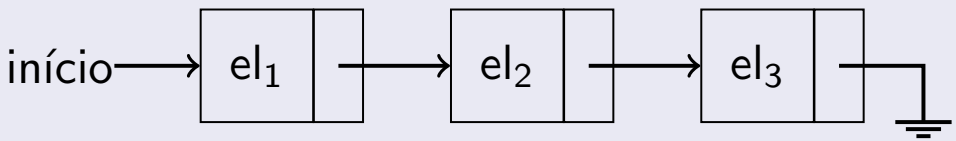
Filas

- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?
  - Quem será o primeiro a ser atendido?
- Se queremos retirar um elemento, retiramos da frente; se queremos incluir, incluimos no final da fila

Filas

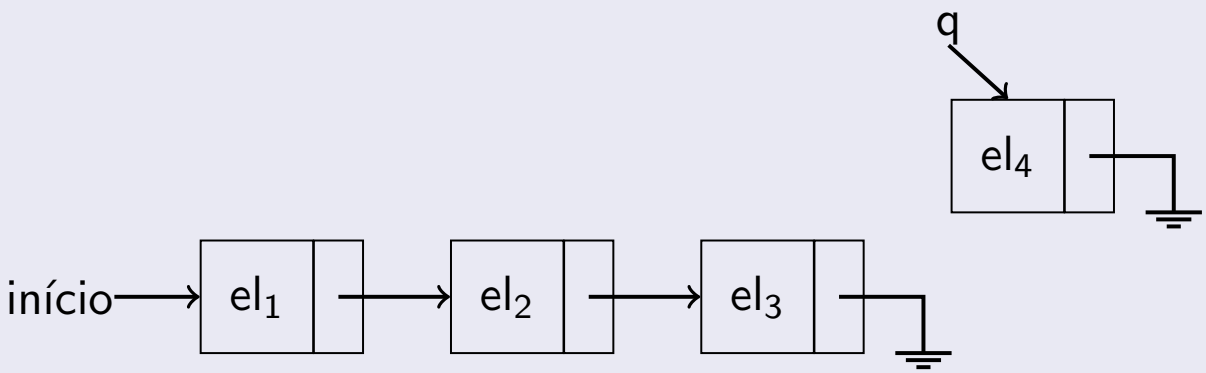
- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?
  - Quem será o primeiro a ser atendido?
- Se queremos retirar um elemento, retiramos da frente; se queremos incluir, incluimos no final da fila
- Da mesma forma que em uma pilha, em uma fila não podemos retirar um elemento do meio da fila, ou lá colocar um

Incluindo um Elemento



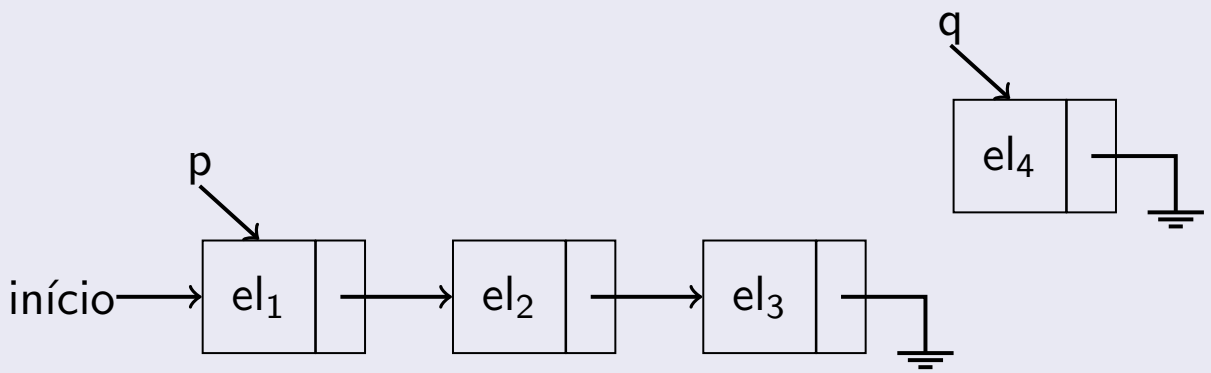
Incluindo um Elemento

- Criamos o elemento novo



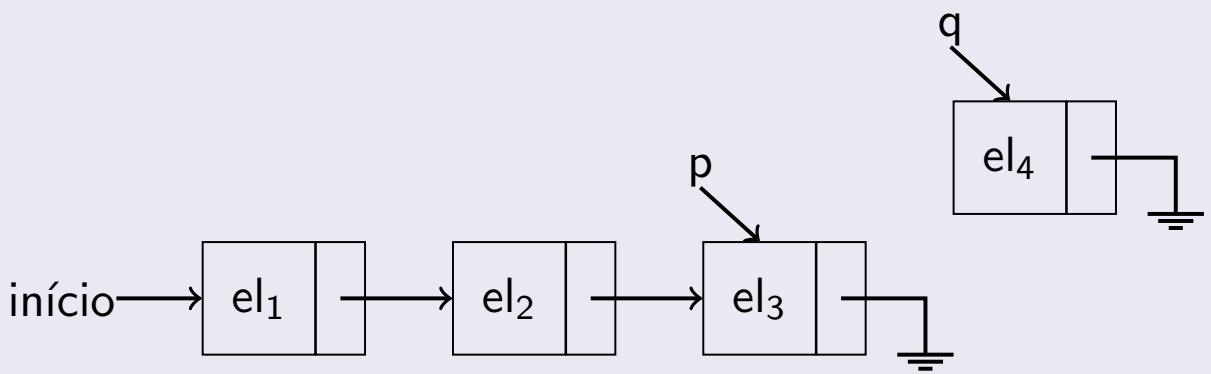
Incluindo um Elemento

- Criamos o elemento novo
- Marcamos o início da fila



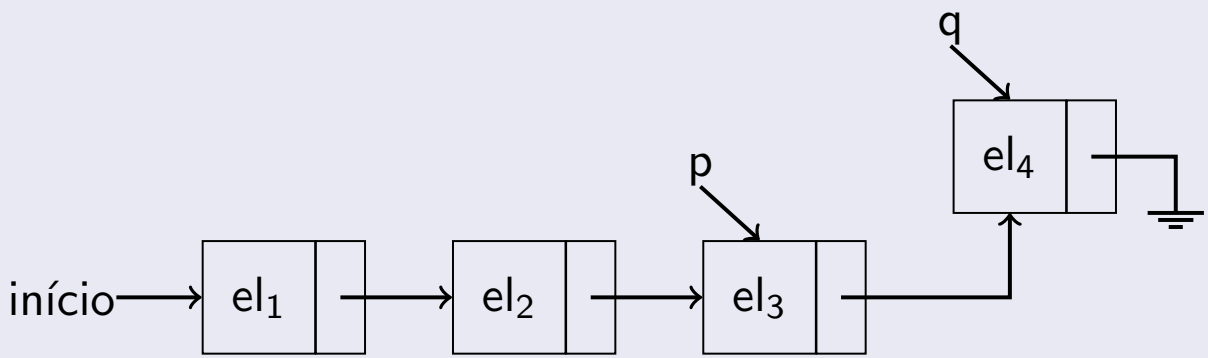
Incluindo um Elemento

- Corremos o ponteiro até o final da fila



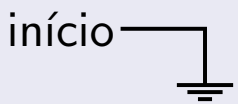
Incluindo um Elemento

- Corremos o ponteiro até o final da fila
- Fazemos o elemento seguinte ao final da fila ser o novo elemento



Incluindo um Elemento

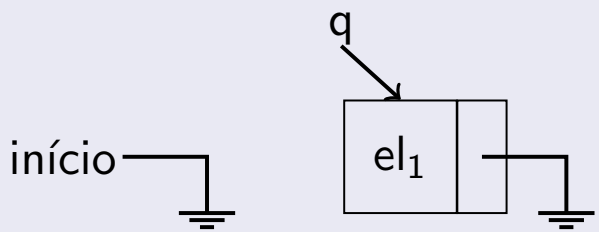
- E se a fila estiver inicialmente vazia?



Filas

Incluindo um Elemento

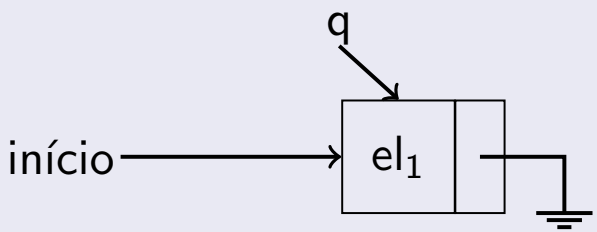
- E se a fila estiver inicialmente vazia?
- Criamos o novo elemento



Filas

Incluindo um Elemento

- E se a fila estiver inicialmente vazia?
- Criamos o novo elemento
- E fazemos ele ser o início da fila



Excluindo um Elemento

- E como excluímos de uma fila?

Filas

Excluindo um Elemento

- E como excluímos de uma fila?
- Do mesmo modo que em uma pilha

Aula 30 – Listas Ligadas

Norton T. Roman & Luciano A. Digiampietri