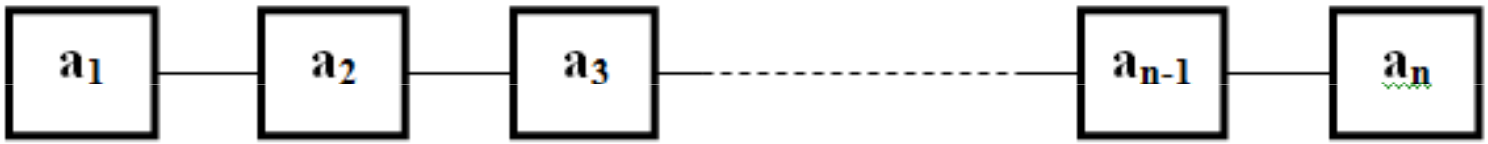
CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

DEFINIÇÃO

- *Lista linear* é uma sequência de zero ou mais elementos de um mesmo tipo.
- Simbolicamente, L = a_1, a_2, ..., a_n, onde n ≥ 0.

L



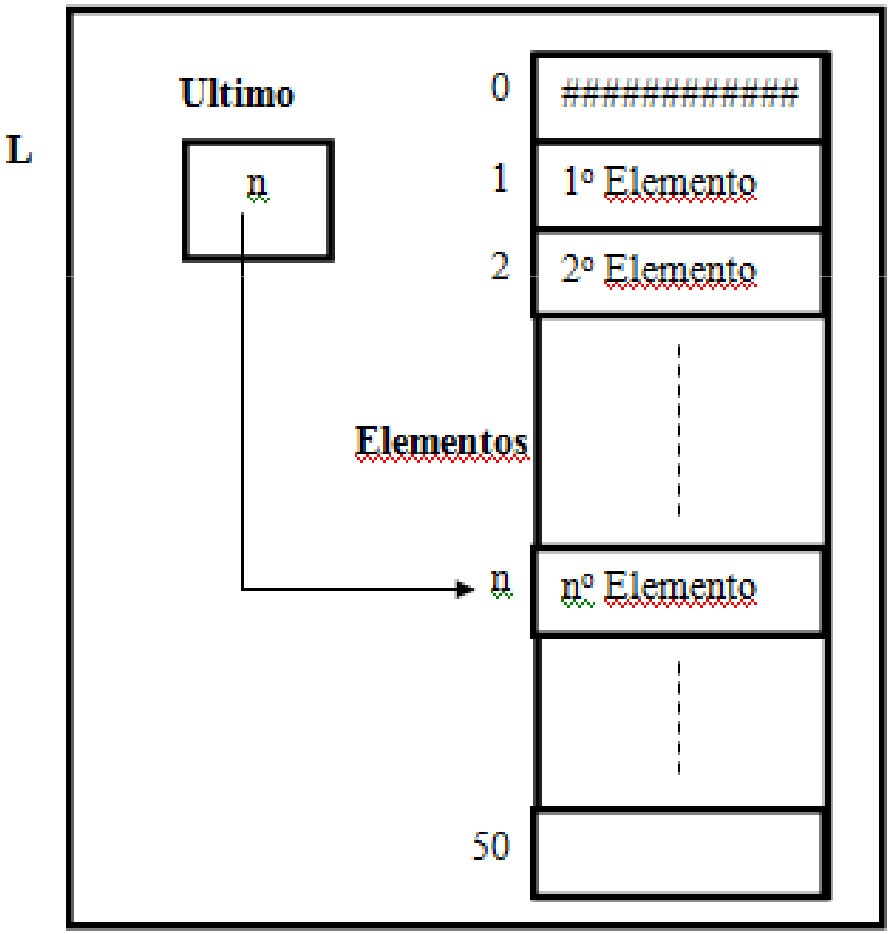
- **Suas posições obedecem uma ordem: a_i precede a_{i+1}, onde 0 < i ≤ n-1.**
- **Seus elementos podem ser consultados, inseridos ou eliminados em qualquer posição.**
- **Listas lineares distintas podem ser concatenadas ou repartidas.**

CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

IMPLEMENTAÇÃO COM VETOR

- Um exemplo: lista linear de caracteres.



Permite a posição 0 desocupada

```
const int max = 50;

typedef char vetor[max+1];

struct lista {
        vetor Elementos;
        int Ultimo;
};

lista L1, L2, L3;
```

IMPLEMENTAÇÃO COM VETOR - CRIAÇÃO

```
lista criarLista () {
   lista L; int i;
   printf("Digite o numero de elementos: ");
   scanf("%d",&L.Ultimo);
   if (L.Ultimo > max) {
        printf("Numero excede tamanho maximo para listas");
        L.Ultimo = 0;
   }
   else if (L.Ultimo > 0) {
        printf ("Digite %d elemento(s): ", L.Ultimo);
        for (i = 1; i <= L.Ultimo; i++)
               scanf("%c",& L.Elementos[i]);
   }
   return L;
}
```

```
void main () {
      lista L1;
      L1 = criarLista ();
}
```

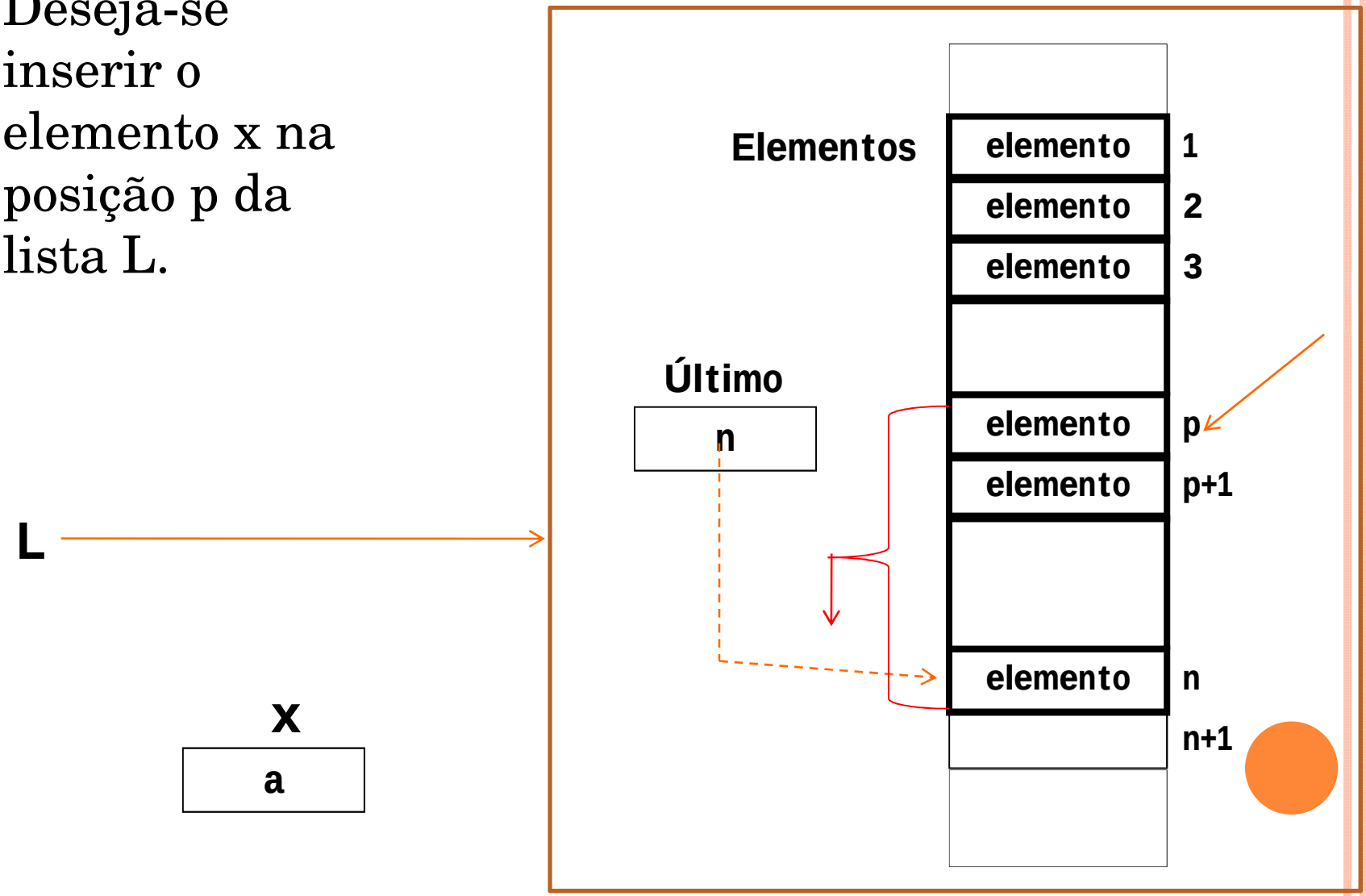
IMPLEMENTAÇÃO COM VETOR - IMPRESSÃO

```
void escreverLista (lista L) {
   int i;
   if (L.Ultimo < 1) printf("Lista vazia");
   else
       for (i = 1; i <= L.Ultimo; i++)
             printf("%c",L.Elementos[i]);
}
```

```
void main () {
      lista L1;
      - - - - -
      escreverLista (L1);
}
```

Implementação com vetor - Inserção

- Deseja-se inserir o elemento x na posição p da lista L.



IMPLEMENTAÇÃO COM VETOR - INSERÇÃO

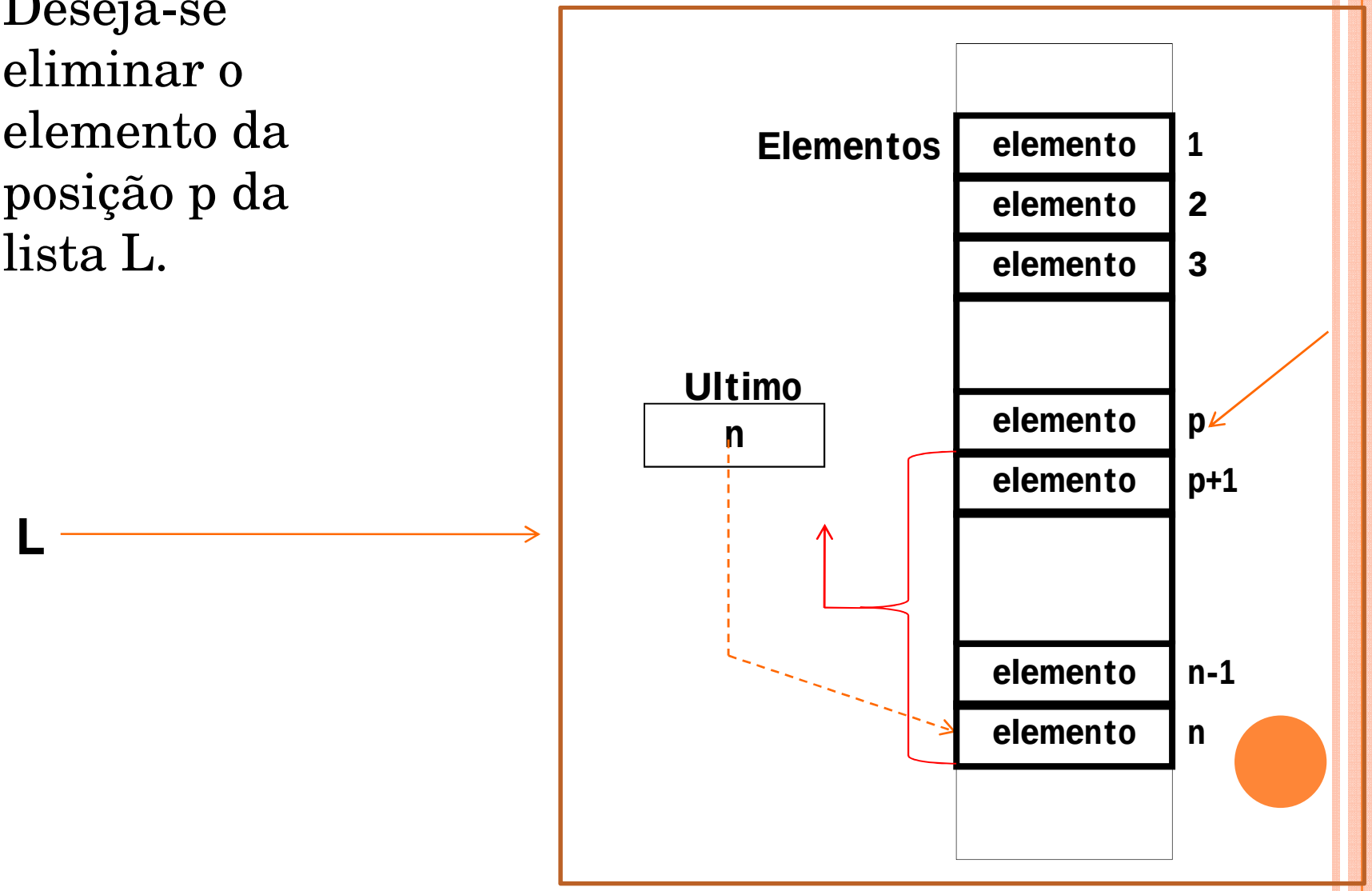
```
void inserirElemento (char x, int p, lista *L) {
   int q;
   if (L->Ultimo >= max)
        printf ("Lista cheia: insercao impossivel");
   else if (p < 1 || p > L->Ultimo + 1)
       printf("A posicao de insercao nao existe");
       else {
              L->Ultimo++;
              for (q = L->Ultimo - 1; q >= p; q--)
                     L->Elementos[q+1] = L->Elementos[q];
              L->Elementos[p] = x;
              }
}
```

No pior caso, tempo gasto será proporcional ao tamanho da lista

Qual a ordem de complexidade? O(?)

Implementação com vetor - Eliminação

- Deseja-se eliminar o elemento da posição p da lista L.



IMPLEMENTAÇÃO COM VETOR - ELIMINAÇÃO

```
void eliminarElemento (int p, lista *L) {
   int q;
   if (p < 1 || p > L->Ultimo)
       printf("A posicao de eliminacao nao existe");
   else {
       L->Ultimo--;
       for (q = p; q <= L->Ultimo; q++)
              L->Elementos[q] = L->Elementos[q+1];
   }
}
```

No pior caso, tempo gasto será proporcional ao tamanho da lista

O(?) ?

Implementação com vetor

- Outras operações:
  - Encontrar em L o p-ésimo elemento
    - `L.Elementos[p]`
  - Encontrar em L o elemento sucessor de p
    - `L.Elementos[p+1]`
  - Encontrar em L o elemento anterior a p
    - `L.Elementos[p-1]`
  - Esvaziar a lista L
    - `L.Ultimo = 0`

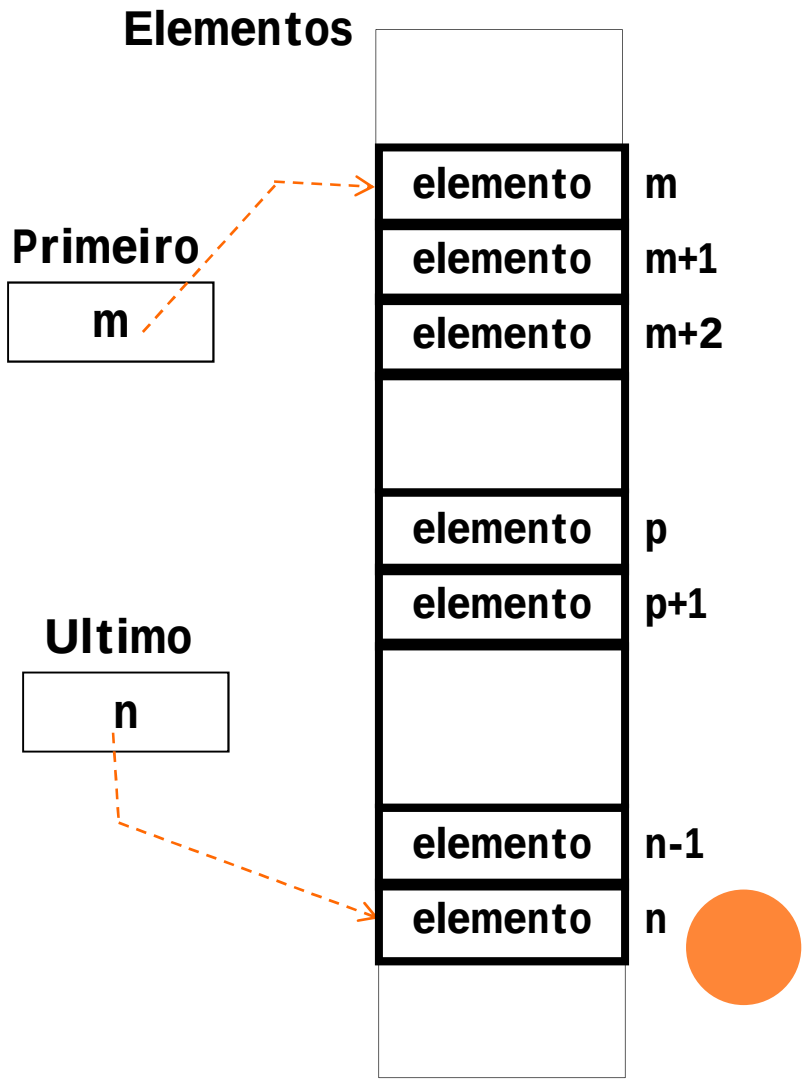
Todas essas operações podem ser realizadas em *tempo constante.*

O(?)

IMPLEMENTAÇÃO COM VETOR

- Uma implementação alternativa

```
const int max = 50;

typedef char vetor[max+1];

struct lista {
       vetor Elementos;
       int Primeiro, Ultimo;
};
```

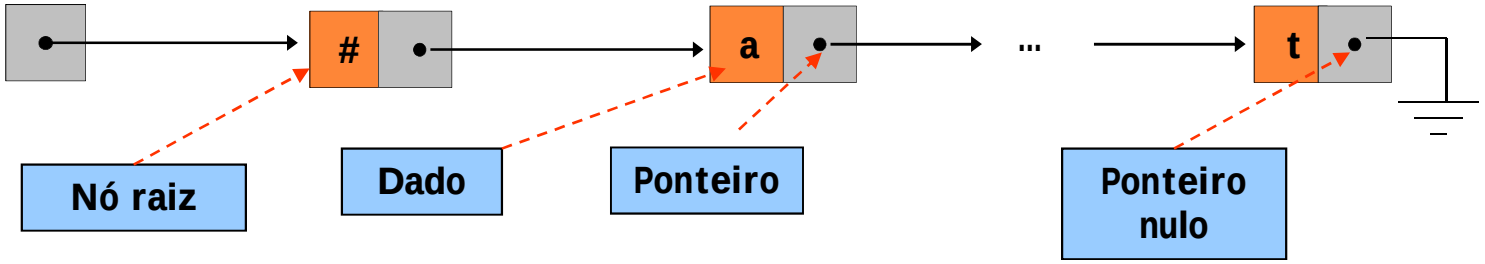


CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

IMPLEMENTAÇÃO COM NÓS ENCADEADOS

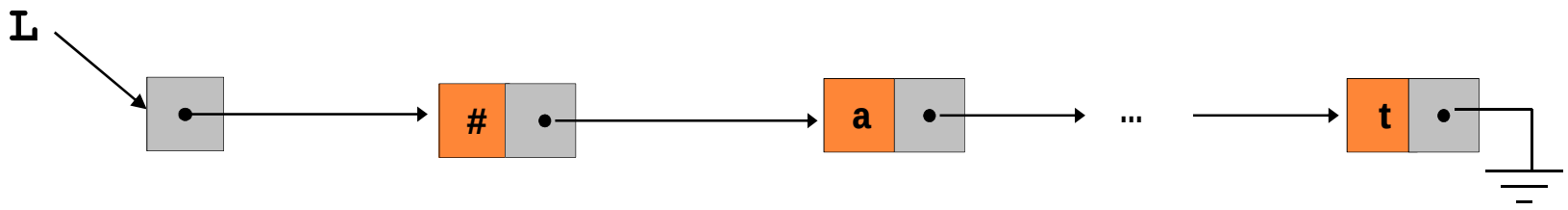
- Os elementos da lista são armazenados num encadeamento de estruturas com ponteiros:



- **Cada uma dessas estruturas:**
  - **recebe o nome de nó;**
  - **armazena um elemento e um ponteiro para o próximo nó.**
- **Para simplificar os códigos, o primeiro nó *costuma* ser definido como raiz, e não armazena nenhum elemento.**
- **Esta estrutura de dados é chamada de lista encadeada ou lista ligada (*linked list*).**

IMPLEMENTAÇÃO COM NÓS ENCADEADOS

- Mesmo exemplo: lista linear de caracteres.

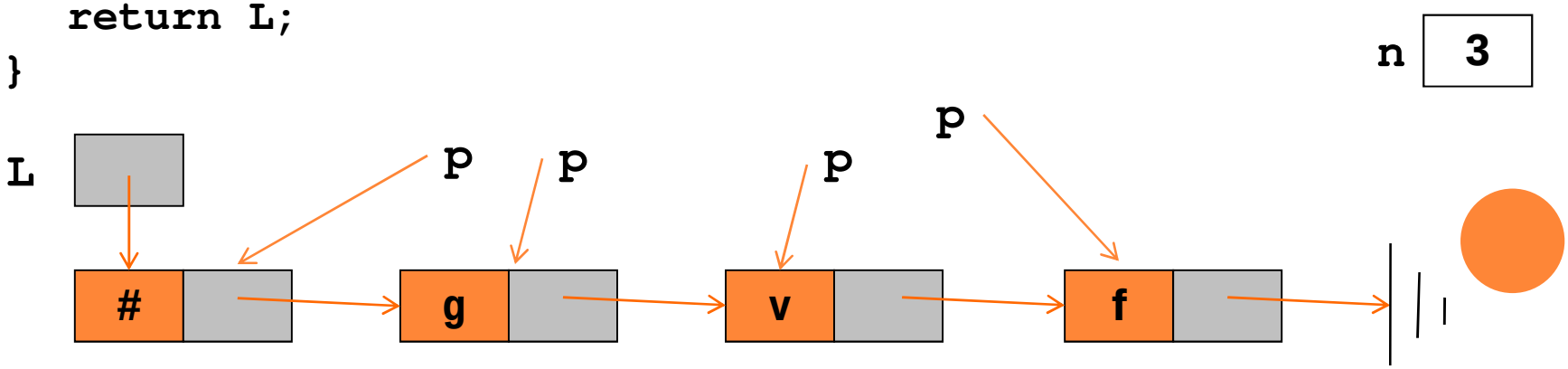



```
struct node {
        char elem;
        node *prox;
};

typedef node *lista;
typedef node *posicao;

lista L1, L2, L3;
L1 = NULL;
posicao p, q, r;
```

Agora lista é um ponteiro!

IMPLEMENTAÇÃO COM NÓS - CRIAÇÃO

```
lista criarLista () {
   int i, n; lista L; posicao p;
   L = (node *) malloc (sizeof (node));
   printf("Digite o numero de elementos: "); scanf("%d",&n);
   if (n > 0) {
       printf ("Digite %d elemento(s): ",n);
        for (p = L, i = 1; i <= n; i++) {
                p->prox = (node *) malloc (sizeof (node));
                p = p->prox; read (p->elem);
        }
    }
   p->prox = NULL;
   return L;
}
```



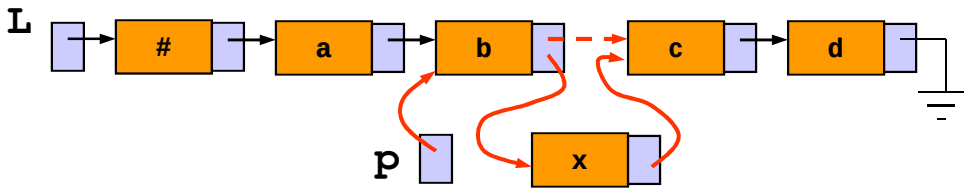
IMPLEMENTAÇÃO COM NÓS - IMPRESSÃO

```
void escreverLista (lista L) {
   posicao p;
   if (L->prox == NULL) printf("Lista vazia");
   else
        for (p = L->prox; p != NULL; p = p->prox)
              printf(" %c ",p->elem);
}
```

```
void main () {
      lista L1;
      - - - - -
      escreverLista (L1);
}
```

IMPLEMENTAÇÃO COM NÓS - INSERÇÃO

- Deseja-se inserir o elemento x após o nó da lista L apontado por p.



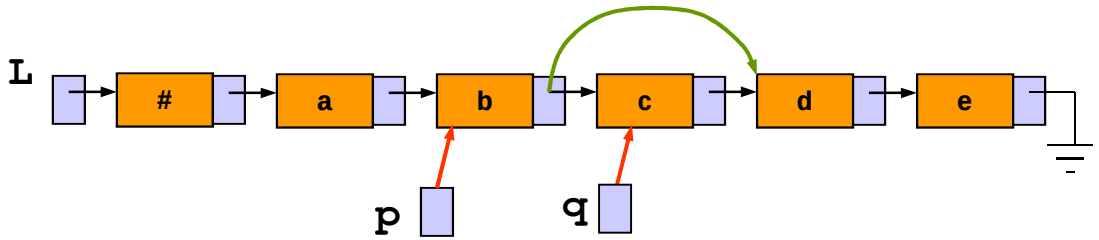

```
void inserirElemento (char x, posicao p, lista L) {
  posicao q;
  if (p == NULL) printf("Posicao nao existe");
  else {
       q = p->prox;
       p->prox = (node *) malloc (sizeof (node));
       p->prox->elem = x;
       p->prox->prox = q;
  }
}
```

Desnecessário, pois p é ponteiro

Inserção pode ser realizada em *tempo constante*

IMPLEMENTAÇÃO COM NÓS - ELIMINAÇÃO

- Deseja-se eliminar o elemento da lista L que está após o nó apontado por p.




```
void eliminarElemento (posicao p, lista L){
   posicao q;
   if (p == NULL || p->prox == NULL)
        printf("Posicao nao existe");
   else {
        q = p->prox;
        p->prox = q->prox;
        free (q);
   }
}
```

Desnecessário, pois p é ponteiro

Eliminação também pode ser realizada em *tempo constante*

IMPLEMENTAÇÃO COM NÓS - ESVAZIAMENTO

```
void esvaziarLista (lista L) {
   posicao p;
   if (L == NULL)
        printf("A lista nao foi inicializada");
   else {
        while (L->prox != NULL) {
              p = L->prox;
              L->prox = L->prox->prox;
              free (p);
        }
   }
}
```

Vai restar apenas o nó lider

```
void main () {
      lista L1;
      - - - - -
      esvaziarLista (L1);
}
```

Implementação com nós encadeados

- Outras operações:
  - Encontrar em L o elemento sucessor do nó apontado por p
    - `p->prox->elem`
  - Encontrar em L o p-ésimo elemento
    - De modo geral, será preciso percorrer a lista
  - Encontrar em L o elemento anterior a p
    - Idem: também será preciso percorrer a lista

As duas últimas operações acima, no pior caso, gastam *tempo linear* em relação ao tamanho da lista

CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

COMPARAÇÃO: TEMPO

- Podemos comparar as duas implementações de listas lineares de n elementos em termos de *tempo de pior caso* na execução das suas principais operações:

Operações	*Vetor*	*Nós encadeados*
Criação	O(n)	O(n)
Impressão	O(n)	O(n)
Inserção	O(n)	O(1)
Eliminação	O(n)	O(1)
Busca	O(n)	O(n)
Esvaziamento	O(1)	O(n)
p-ésimo	O(1)	O(n)
Sucessor	O(1)	O(1)
Anterior	O(1)	O(n) ?

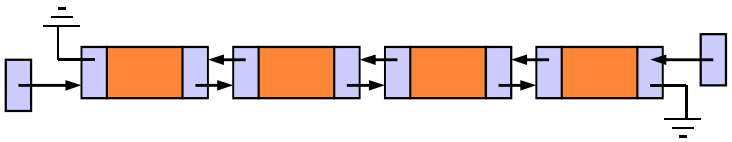
COMPARAÇÃO: ESPAÇO

- Em relação ao gasto de memória:
  - Vetores utilizam espaço constante, o que pode ser desvantajoso para listas pequenas
  - Listas encadeadas exigem espaço extra para o armazenamento de ponteiros
- No caso das listas encadeadas, com um gasto extra de memória (um ponteiro para o nó anterior), é possível tornar constante o tempo de acesso ao elemento anterior.
  - Esta estrutura é chamada de *lista duplamente encadeada*

CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

LISTAS DUPLAMENTE ENCADEADAS



```
struct node {
        char elem;
        node *prox, *prev;
};

typedef node *lista;
typedef node *posicao;
```

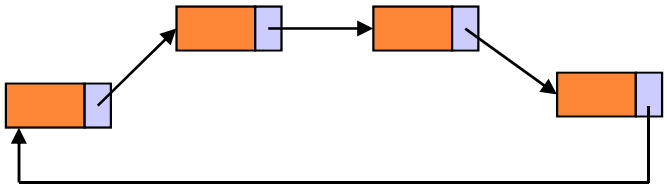
- Também será mantido um ponteiro para o final da lista.
- Vantagem: acesso ao anterior em tempo constante.
- Desvantagens: maior consumo de memória, código um pouco mais complicado (atualização de mais ponteiros).

CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

LISTAS CIRCULARES

- Determinadas aplicações exigem outra variante: uma *lista encadeada circular*.



- **Nesses casos, deixa de haver a noção de primeiro da lista, dispensando-se o uso de um nó líder.**
- **Basta manter um ponteiro L para algum dos elementos da lista.**
- **As listas circulares também podem ser duplamente encadeadas.**

CES-11

- Listas lineares
  - Definição
  - Implementação com vetor
  - Implementação com nós encadeados
  - Comparação
  - Outras implementações
    - Listas duplamente encadeadas
    - Listas circulares
  - Exercícios

EXERCÍCIOS

- Dadas duas listas lineares, deseja-se implementar a operação de *concatenação*, isto é, acrescentar a segunda lista no final da primeira. Qual seria a melhor implementação para realizar essa operação? Há algum modo de torná-la mais eficiente?
- Escreva o código das principais operações (criação, impressão, inserção, eliminação e esvaziamento) em uma:
  - lista encadeada sem nó líder;
  - lista duplamente encadeada;
  - lista encadeada circular (sem nó líder).