CES-11

Algoritmos e Estruturas de Dados

Carlos Alberto Alonso Sanches
Juliana de Melo Bezerra

CES-11

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

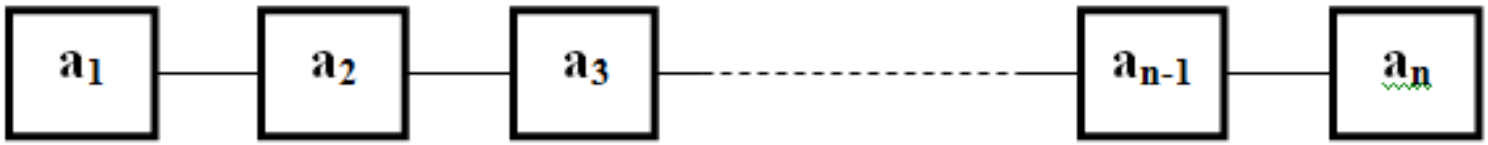
CES-11

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

Definição

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

L



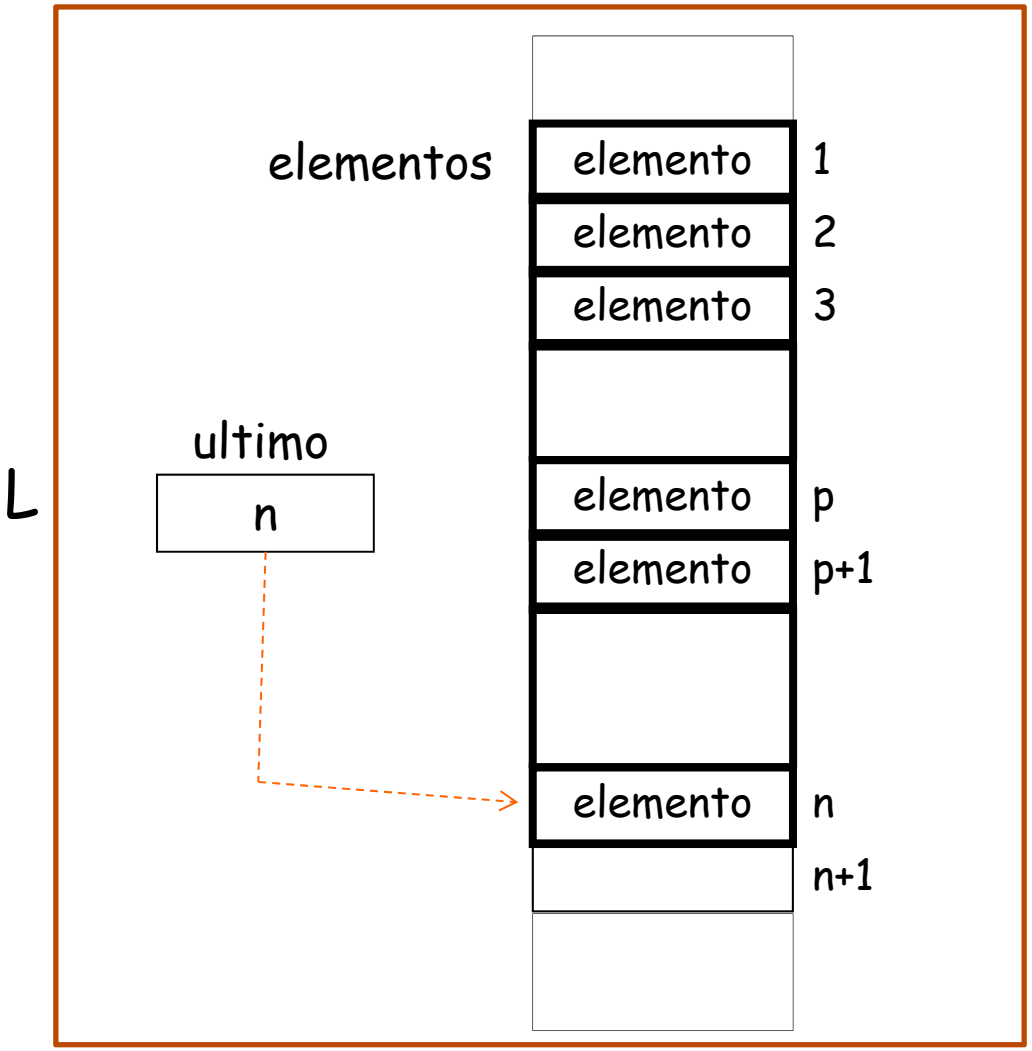
- Suas posições obedecem a uma ordem: a_i precede a_{i+1}, onde $0 < i < n$.
- Seus elementos podem ser consultados, inseridos ou eliminados em qualquer posição.
- Listas lineares distintas podem ser concatenadas ou repartidas.

CES-11

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

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;
   write ("Digite o numero de elementos: ");
   read (L.ultimo);
   if (L.ultimo > max) {
        Erro ("Numero excede tamanho maximo para listas");
        L.ultimo = 0;
   }
   else if (L.ultimo > 0) {
        write ("Digite ", L.ultimo, " elemento(s): ");
        for (i = 1; i <= L.ultimo; i++)
               read (L.elementos[i]);
   }
   return L;
}
```

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

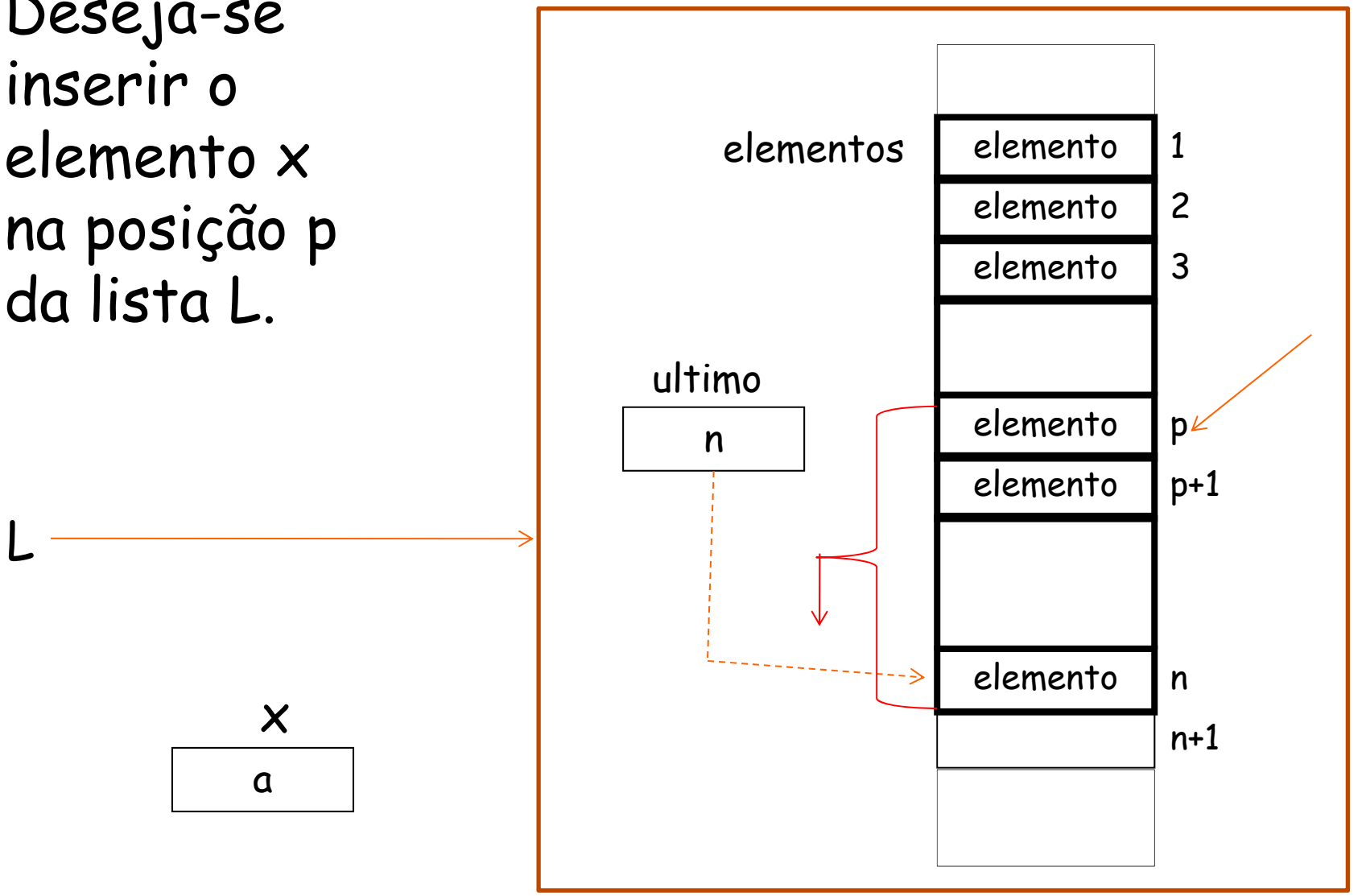
Implementação com vetor - Impressão

```
void escreverLista (lista L) {
   int i;
   if (L.ultimo < 1) write ("Lista vazia");
   else
        for (i = 1; i <= L.ultimo; i++)
              write (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.



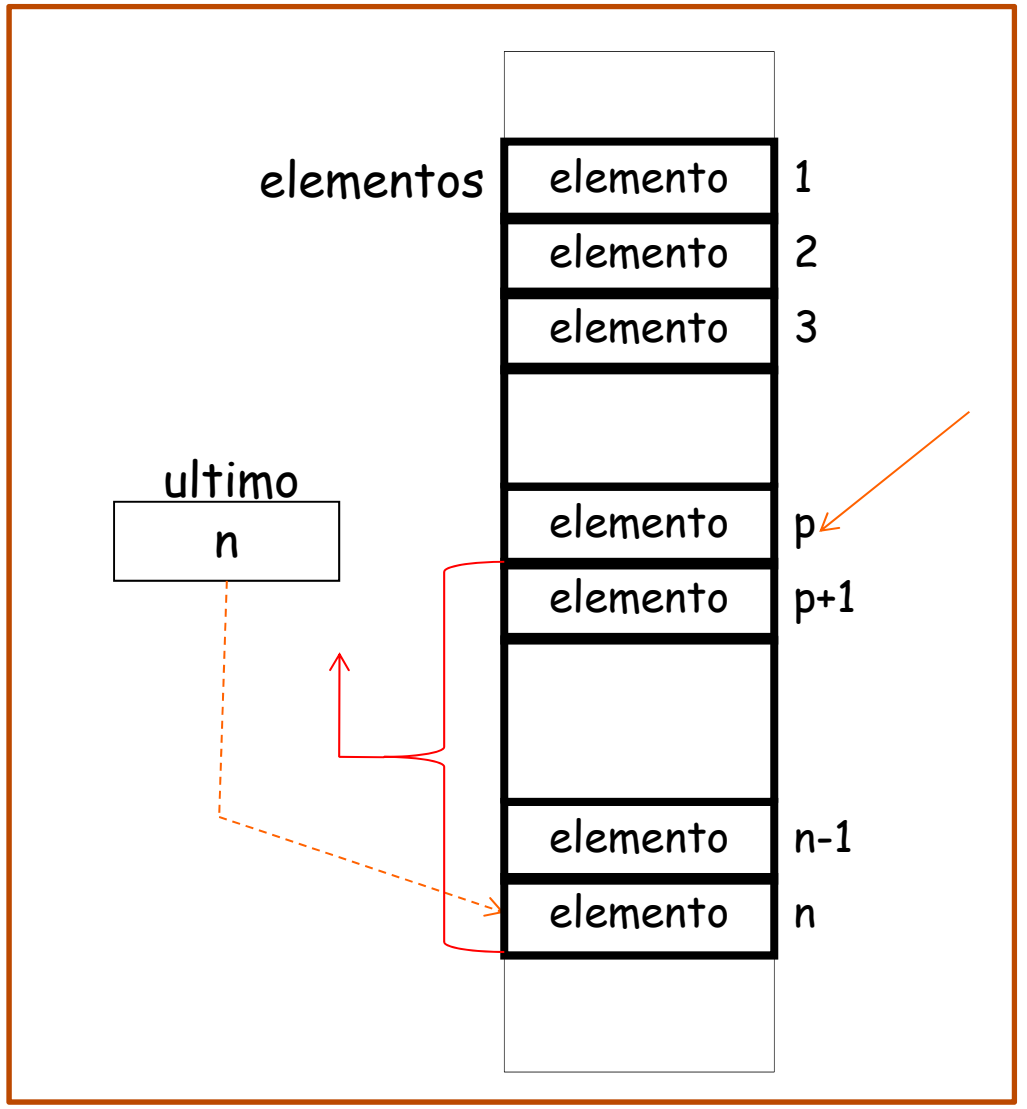
Implementação com vetor - Inserção

```
void inserirElemento (char x, int p, lista *L) {
   int q;
   if (L->ultimo >= max)
       Erro ("Lista cheia: insercao impossivel");
   else {
       if (p < 1 || p > L->ultimo + 1)
              Erro ("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

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)
        Erro ("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

Implementação com vetor - Busca

- Deseja-se encontrar a posição da primeira ocorrência do elemento x na lista L.

```
int buscarElemento (char x, lista L) {
   int posic = -1, q = 1;
   bool achou = false;
   while (q <= L.ultimo && !achou) {
       if (L.elementos[q] != x)
          q++;
       else {
          posic = q;
          achou = true;
       }
    }
    return posic;
}
```

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

Implementação com vetor

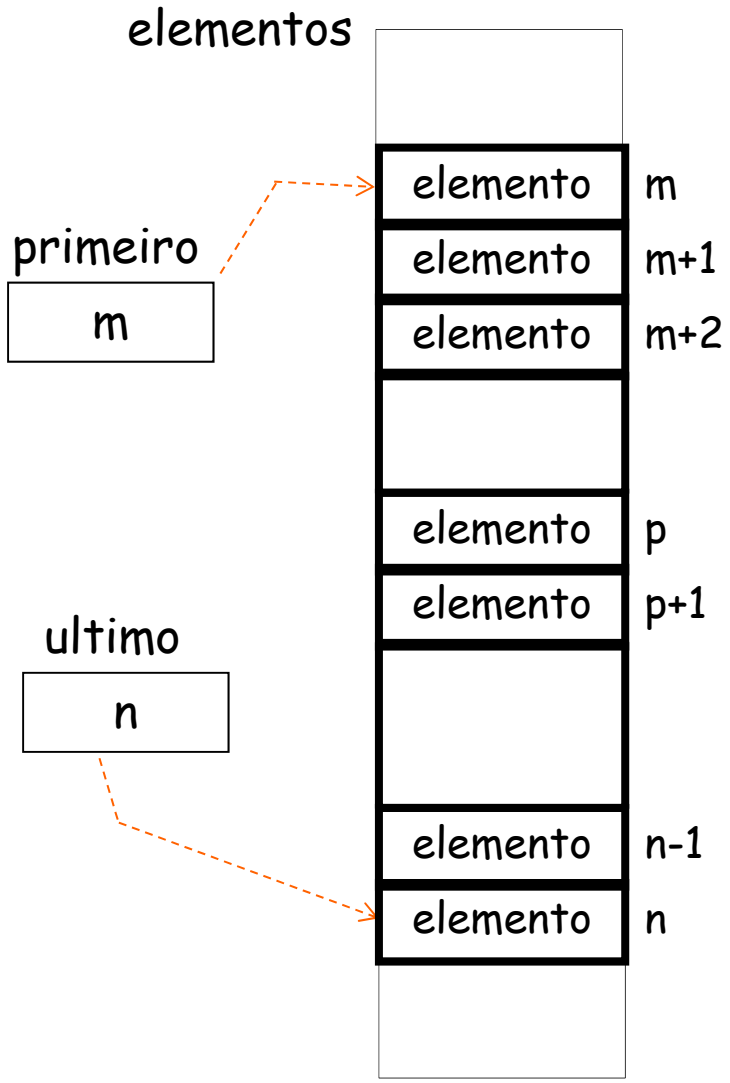
- Outras operações:
  - Encontrar em L o p-ésimo elemento
    - `L->elementos[p]`
  - Encontrar em L o elemento sucessor de p
    - `L->elementos[p+1]`
  - Encontrar em L o elemento anterior a p
    - `L->elementos[p-1]`
  - Esvaziar a lista L
    - `L->ultimo = 0`

Todas essas operações podem ser realizadas em *tempo constante*

Implementação com vetor

- Uma implementação alternativa

```
const int max = 50;

typedef char vetor[max+1];

struct lista {
       vetor elementos;
       int primeiro, ultimo;
};
```



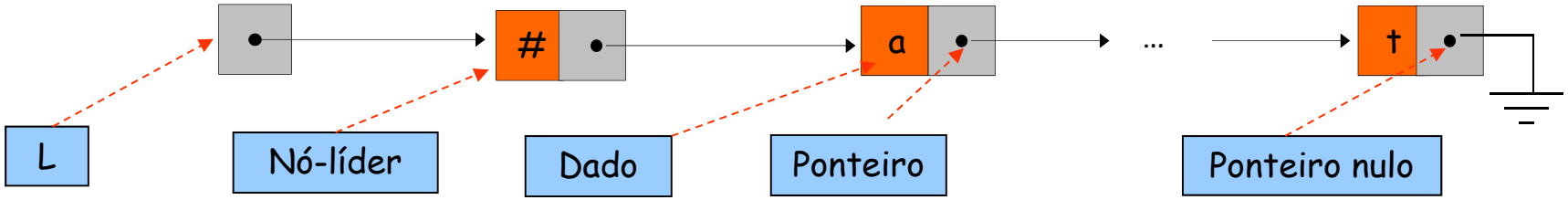
Vantagem: uso do vetor de forma circular

CES-11

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

Implementação com nós encadeados

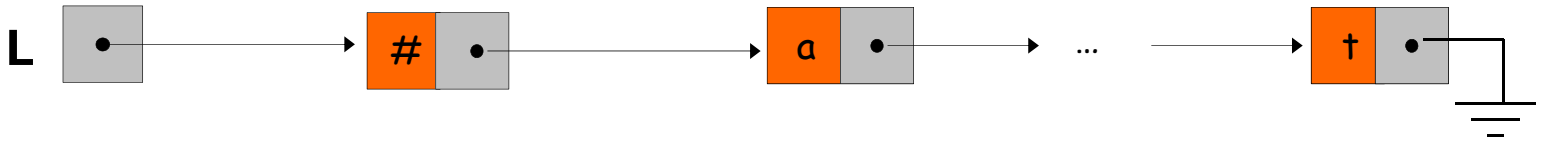
- Os elementos da lista são armazenados em um encadeamento de estruturas com ponteiros:



- Cada uma dessas estruturas:
  - recebe o nome de nó;
  - armazena um elemento e um ponteiro para o próximo nó.
- O primeiro nó *pode* ser definido como líder (também chamado de *sentinela*), não armazenando nenhum elemento.
  - Desse modo, inserções e remoções na lista não alteram valor do ponteiro L.
  - Codificação torna-se mais simples.
- Esta estrutura de dados é chamada de lista encadeada ou lista ligada (*linked list*).

Implementação com nós encadeados

- Mesmo exemplo: lista linear de caracteres.

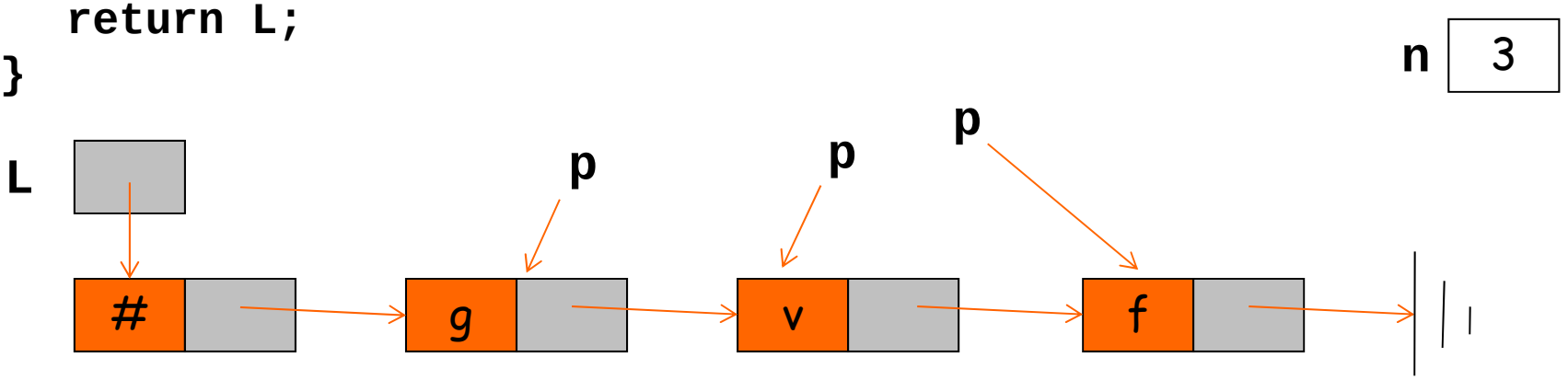



```
struct node {
        char elem;
        node *prox;
};

typedef node *lista;
typedef node *posicao;

lista L1, L2, L3;
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)); p = L;
   write ("Digite o numero de elementos: "); read (n);
   if (n > 0) {
       write ("Digite ", n, " elemento(s): ");
       for (i = 1; i <= n; i++) {
              p->prox = (node *) malloc (sizeof (node));
              p = p->prox; read (p->elem);
       }
    }
    p->prox = NULL;
    return L;
}
```



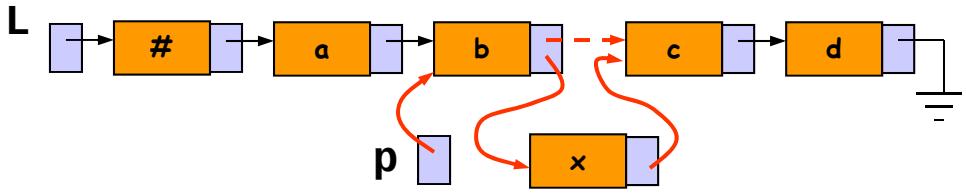
Implementação com nós - Impressão

```
void escreverLista (lista L) {
   posicao p;
   if (L->prox == NULL) write ("Lista vazia");
   else
       for (p = L->prox; p != NULL; p = p->prox)
              write (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.



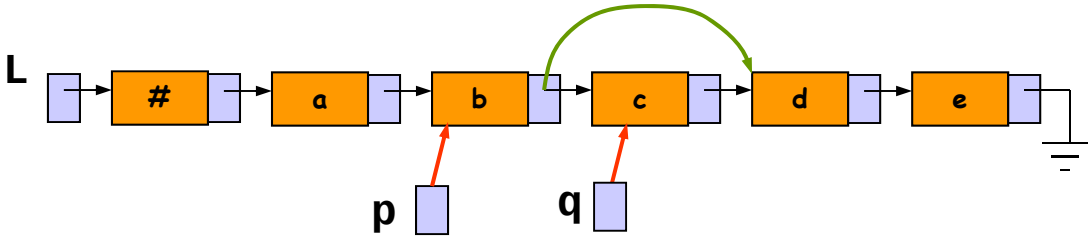

```
void inserirElemento (char x, posicao p, lista L) {
  posicao q;
  if (p == NULL) Erro ("Posicao nao existe");
  else {
       q = p->prox;
       p->prox = (node *) malloc (sizeof (node));
       p->prox->elem = x;
       p->prox->prox = q;
  }
}
```

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)
        Erro ("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)
        Erro ("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 líder

```
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

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) ?

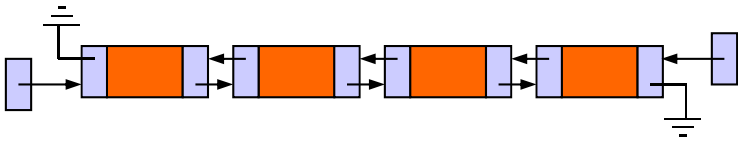
Comparação: espaço

- 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

Listas duplamente encadeadas



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

typedef node *posicao;

struct lista {
        posicao inic, final;
};
```

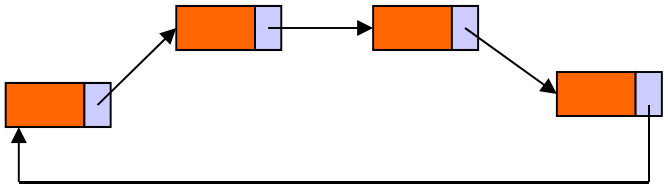
- Também será mantido um ponteiro para o final da lista.
- Vantagem: acesso ao anterior em tempo constante.
- Desvantagens: maior consumo de memória, código um pouco mais complicado (atualização de mais ponteiros).

CES-11

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

Listas circulares

- Há outra variante chamada *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.