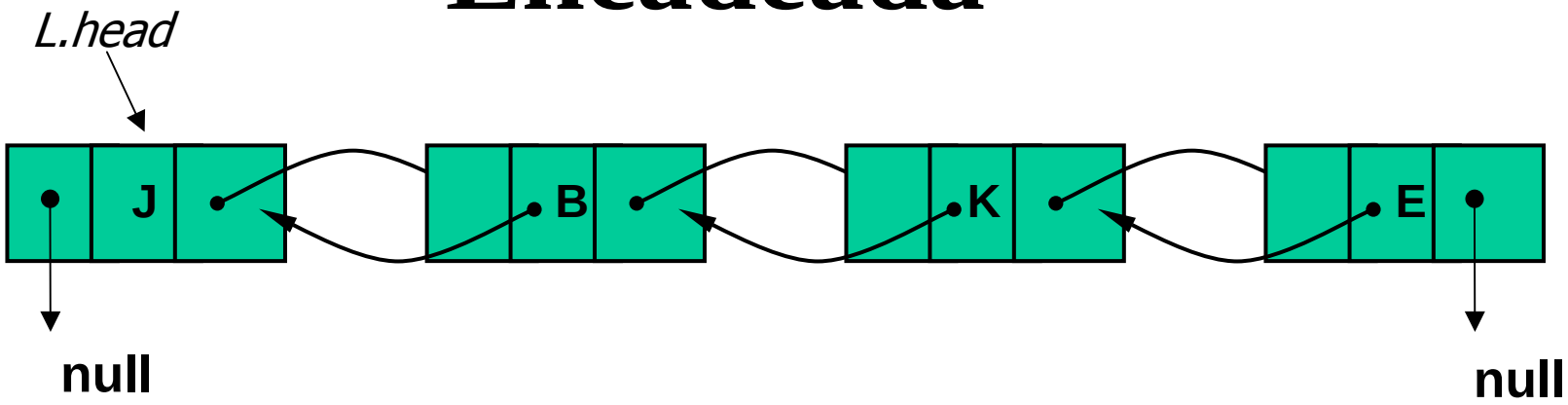
Listas Duplamente Encadeadas

Lista Dinâmica Duplamente Encadeada

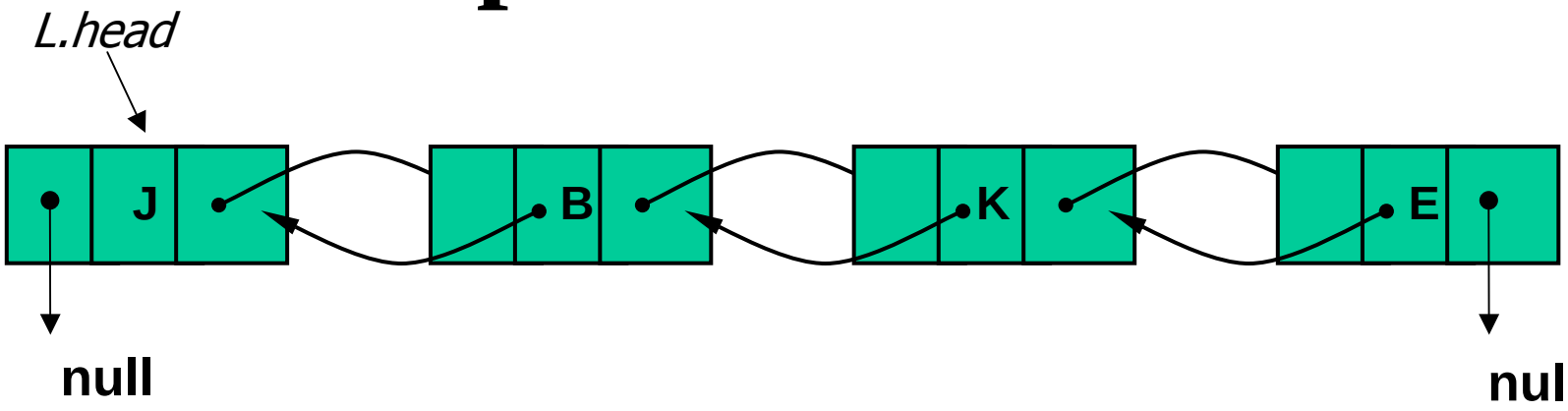


Lista Dinâmica

- **Definição da ED**

```
struct list_rec {
  tipo_elem elem;
  struct list_rec *ligdir, *ligesq;
};

typedef struct list_rec  Rec;
```

Listas Duplamente Encadeadas



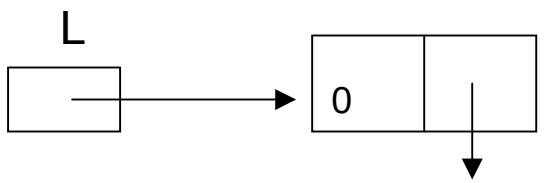
- **Lista:**

```
typedef struct {
        int nelem;
        Rec *head;
} Lista;

Lista *L;   /* Exemplo de Declaração */
```

Implementação das Operações

1) **Criação da lista vazia**

```
void CriarLista(Lista *L){
     L = malloc(sizeof(Lista));
     L->nelem = 0;
     L->head = NULL;
}
```



Implementação das Operações

2) **Inserção do primeiro elemento**

```
void Insere_Prim(Lista *L, Tipo_elem elem){

  Rec *p;
  p = malloc(sizeof(Rec));
  p->elem = elem;
  p->ligdir = NULL;
  p->liesq = NULL;

  L->head = p;
  L->nelem++;
}
```

Implementação das Operações

3) **Inserção no início de uma lista**

```
void Insere_Inicio(Lista *L,
  Tipo_elem elem){
    Rec *p;
    p = malloc(sizeof(Rec));
    p->elem = elem;
    p->ligdir = L->head;
   (L->head)->ligesq = p;
    p->ligesq = null;
    L->head = p;
    L->nelem++;
}
```

Implementação das Operações

4) **Acesso ao primeiro elemento da lista**

```
Tipo_elem Primeiro(Lista *L){
    return L->head->elem;
}
```

Implementação das Operações

Quantos elementos tem a lista ?

```
int Nelem(Lista *L){
    return L->nelem;
}

/* se nelem tiver atualizado */
```

```
int Nelem(Lista *L){

    Rec *p = L->head;
    int count = 0;

    while (p != NULL){
        count ++;
        p = p->ligdir;
    }

    return count;
}
```

Implementação das Operações

versão recursiva

```
int Nelem_rec(Rec *p){

    if (p == NULL)
      return 0;
    else
      return 1 + Nelem_rec(p->ligdir);
}

int Nelem_rec_init(Lista *L){
    return Nelem_rec(L->head);
}
```

Implementação das Operações

5 (a) Buscar registro de chave x em lista ordenada – versão iterativa

```
Boolean Buscar_ord (Lista *L, Tipo_chave x, Rec *p){
/*Busca por x e retorna TRUE e o endereço (p) de x numa Lista
   Ordenada, se achar; senão,retorna FALSE */

    if(L->nelem == 0) /*Lista vazia, retorna NULL*/

        return FALSE;

    else{

        p = L->head;

    /*...*/
```

```
    while (p != NULL){ /* enquanto não achar o final */

            if (p->elem.chave >= x){
                if (p->elem.chave == x)/* achou o registro*/
                    return TRUE;
                else
                    /* achou um registro com chave maior*/
                    return FALSE;
            }else{
                p = p->ligdir;
            }
        }

        /* achou final da lista*/
        return FALSE;
    }
}
```

Implementação das Operações

5 (b) Buscar registro de chave x em lista ordenada (Versão Recursiva)

```
Boolean Busca_ord_rec_init(Lista *L, Tipo_chave x,Rec *p){
/*Busca por x e retorna TRUE e o endereço (p) de x numa
   Lista Ordenada, se achar; senão,retorna FALSE */

    if(L->nelem == 0) /*Lista vazia, não achou*/
        return FALSE;

     p = L->head;
    return Busca_ord_rec(p, &x);

}
```

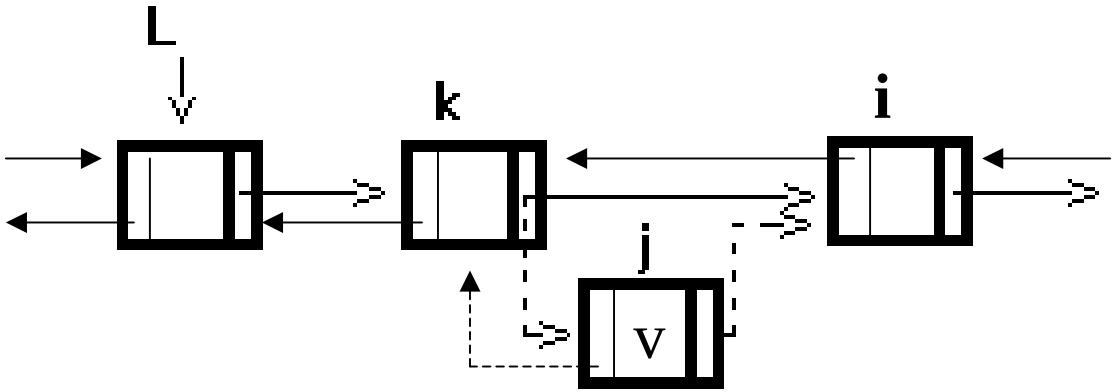
Passagem por endereço, mas poderia ser por valor (economiza espaço)

Exercício: Implementar uma versão para Lista não ordenada!

```
Boolean Busca_ord_rec(Rec *q, Tipo_chave *x){

        if (q == NULL)
            /* chegou no final da lista, sem achar*/
            return FALSE;
        else
            if (q->elem.chave >= *x){
                if (q->elem.chave == *x)
             /* achou o registro*/
                    return TRUE;
                else
                    /* achou um registro com chave maior*/
                    return FALSE;
            }else
                return Busca_ord_rec(q->ligdir, x);
}
```

Implementação das Operações

6) Inserção de elemento *v* como antecessor do elemento no endereço i



Implementação das Operações

6) (a) Inserção de elemento *v* como antecessor do elemento no endereço i

```
void Insere_Antes(Lista *L,Tipo_elem v, Rec *i){
    /* i não pode ser null*/
    Rec *j = malloc(sizeof(Rec));
    j->elem = v;
    j->ligdir = i;
    j->ligesq = i->ligesq;
    if j->ligesq != null
      (j->ligesq)->ligdir = j;
    i->ligesq = j;
    L->nelem++
}
```

Implementação das Operações

(b) Inserção do elemento *v* na lista ordenada L

```
boolean Insere(Lista *L, Tipo_elem v){
/*Insere item de forma a manter a lista ordenada.
Retorna true se inseriu; false, se não foi possível
   inserir*/

    if (L->nelem == 0){
        /*insere como primeiro elemento*/
        insere_Prim(L, v);
        return TRUE;
    }

    Rec *p = L->head;


    /*...*/
```

```
while (p != NULL){

    if (p->elem.chave >= v.chave){
        if (p->elem.chave == v.chave)
            /* v já existe na lista*/
            return FALSE;
        else {
            if (p == L->head)
                /*insere no inicio */
                Insere_Inicio(L, v);
            else{
                /*insere no meio*/
                Insere_Antes(L, v, p);
                }
            return TRUE;
        }
    }else{
         p = p->ligdir;
    }
}                          /*...*/
```

```
    /*insere no final*/
    /* Exercício: como inserir o valor maior que
  todos???? */
    return TRUE;
}
```

Implementação das Operações

(c) Inserção do elemento *v* na lista ordenada L (Recursivo)

Faça como Exercício!!!

Implementação das Operações

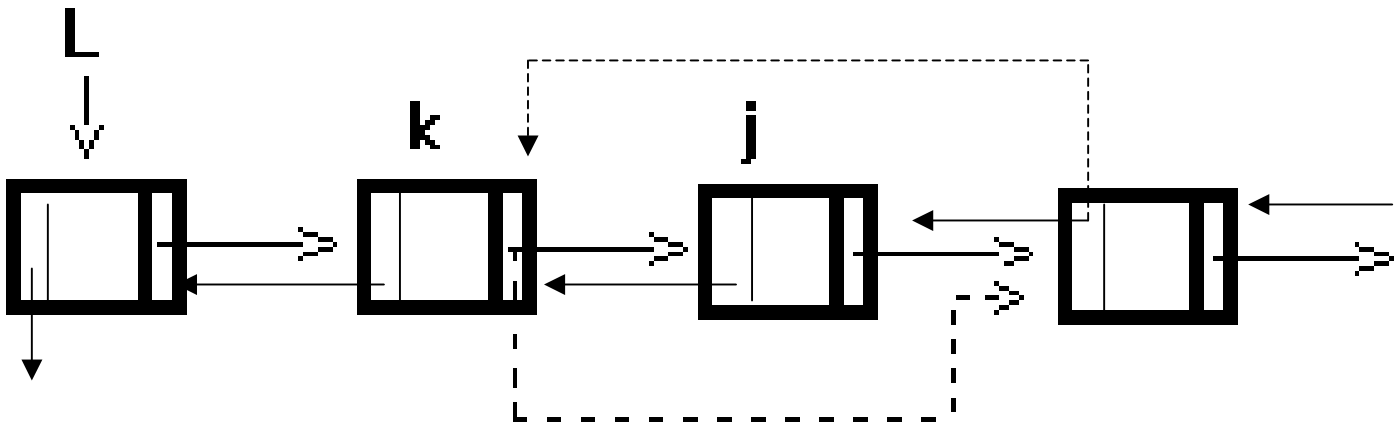
7) **Remoção do primeiro elemento**

```
void Remove_Prim(Lista *L){
  /* supõe que a Lista não está vazia */
    Rec *p = L->head;

    L->head = p->ligdir;
    if (L->head != null)
       (L->head)->ligesq = null;
    free(p);

    L->nelem--;
}
```

Obs: funciona no caso de remoção em lista com um único elemento?

Implementação das Operações

8) **Remoção do elemento apontado por j**



Implementação das Operações

8) **Remoção do elemento apontado por j**

```
void Elimina(Lista *L, Rec *j){

    if (j->ligesq) != null
      (j->ligesq)->ligdir = j->ligdir;
    else /* j é o head => mudar head */
       L->head = j->ligdir;
    if (j->ligdir) != null
       (j->ligdir)->ligesq = j->ligesq;
    /* senão, j é o último; nada a fazer */
    free(j);
    L->nelem--;
}
```

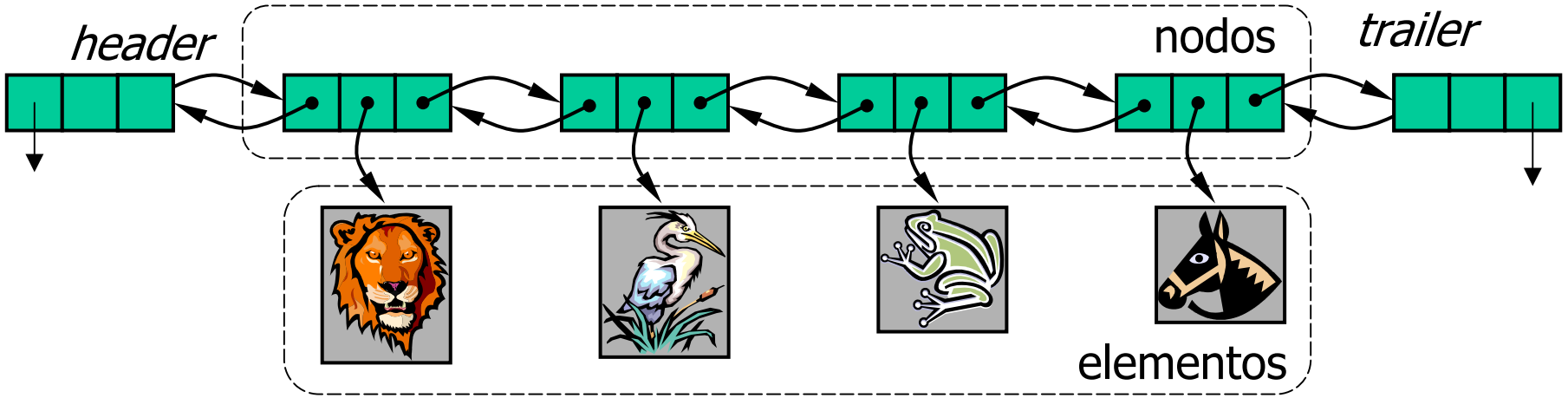
Implementação das Operações

9) **Eliminar elemento *v* de uma lista ordenada L**

```
boolean Remove(Tipo_elem v, Lista*L){
    Rec *p = L->head;
    while (p != NULL){
       if (p->elem.chave < v.chave){
              p = p->lig;
        } else {
            if (p->elem.chave > v.chave)
                /* encontrou elemento com chave maior*/
                return FALSE;
            else {
                /*encontrou o elemento*/
                Elimina(L,p);}
                return TRUE;
           }
       }
    }
    /*não encontrou o elemento na lista*/
    return FALSE;
}
```

Variantes

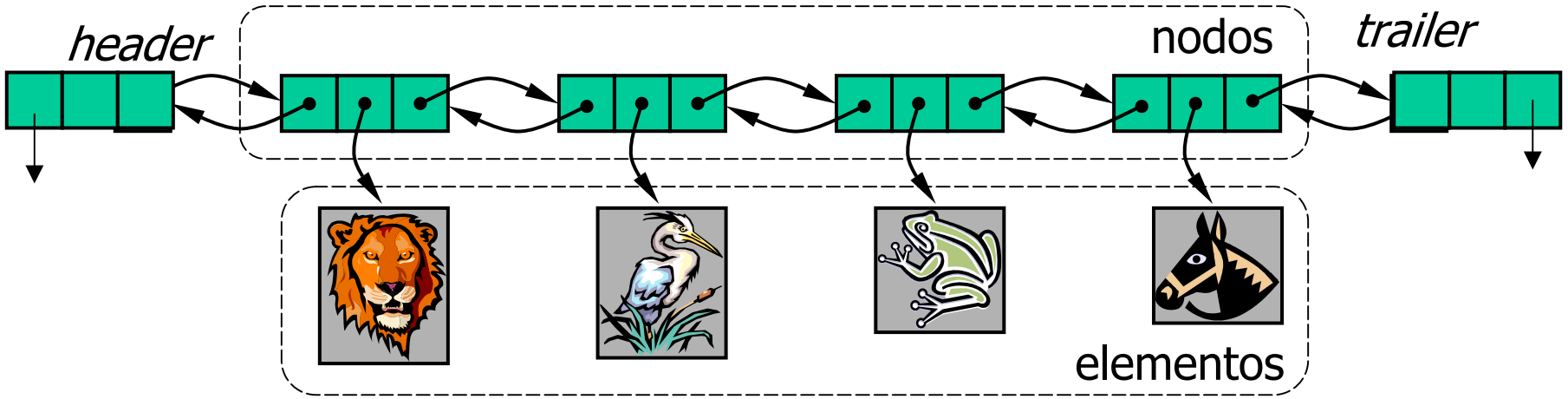
- Lista com **Sentinelas**:
  - Nó especiais *header* e *trailer*
    - Não armazenam elementos
    - *header* possui campo *lig_prev* nulo (**NULL**)
    - *trailer* possui campo *lig_next* nulo (**NULL**)



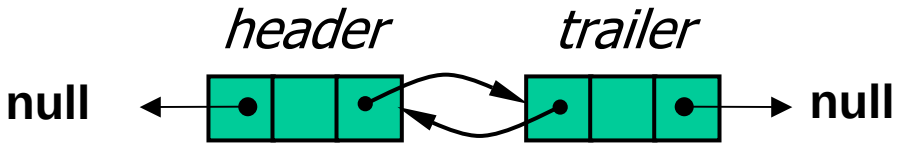
Variantes

- **Lista com Sentinelas:**

```
typedef struct {
        int nelem;
        nodo header, trailer;
      } Lista;
```



Variantes



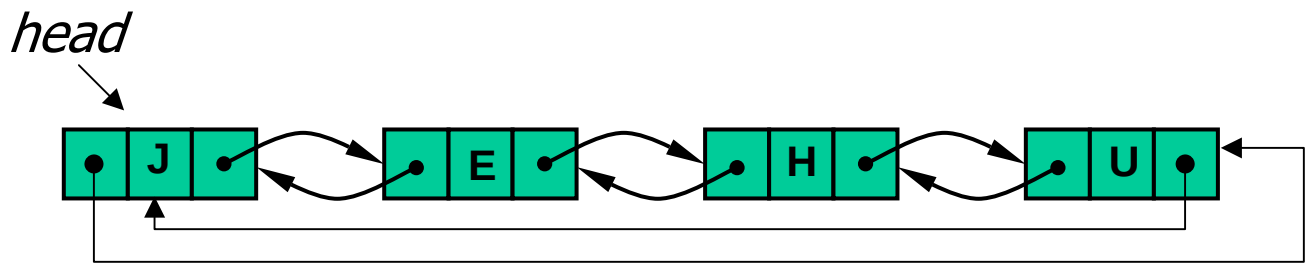
- Lista com Sentinelas (Inicialização):

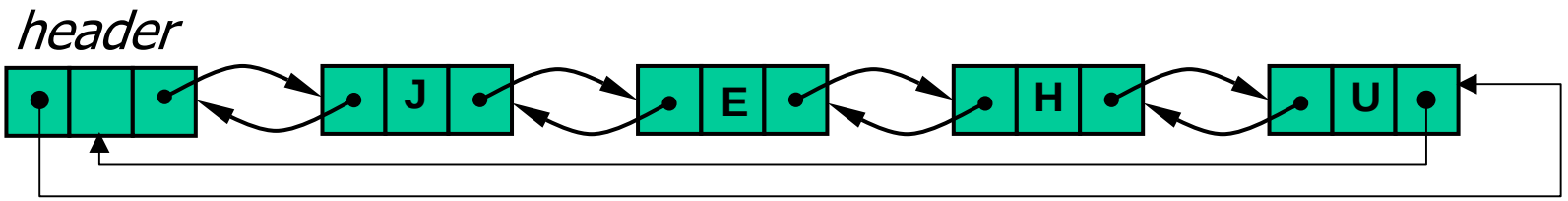
```
Lista *Definir(void){
   Lista *L = malloc(sizeof(Lista));
   L->nelem = 0;
   (L->header).lig_prev = NULL;
   (L->header).lig_next = &(L->trailer);
   (L->trailer).lig_next = NULL;
   (L->trailer).lig_prev = &(L->header);
  return L;
}
```

- Inserção e Remoção:
  - Dispensa diferenciar entre nós intermediários e extremos

Variantes

- Lista **Circular**:
  - Último nodo aponta para o primeiro e vice-versa
  - Pode ser com sentinela ou não





Resumo (Listas Estáticas vs Dinâmicas)

- **Desvantagens**:
  - **Listas Estáticas (Sequenciais)**:
    - Inserir/remover elem. intermediários requer deslocamentos (***O***(***n***))
    - Exige previsão de espaço
  - **Listas Dinâmicas Simplesmente Encadeadas**:
    - Busca (***O***(***n***))
    - Remoção e Inserção requerem cuidado para acesso ao antecessor
    - Acesso por colocação (*rank*) na lista não é direto (***O***(***n***))
  - **Listas Dinâmicas Duplamente Encadeadas**:
    - Acesso por colocação (*rank*) na lista não é direto (***O***(***n***))
    - Duplica o número de campos de ligação