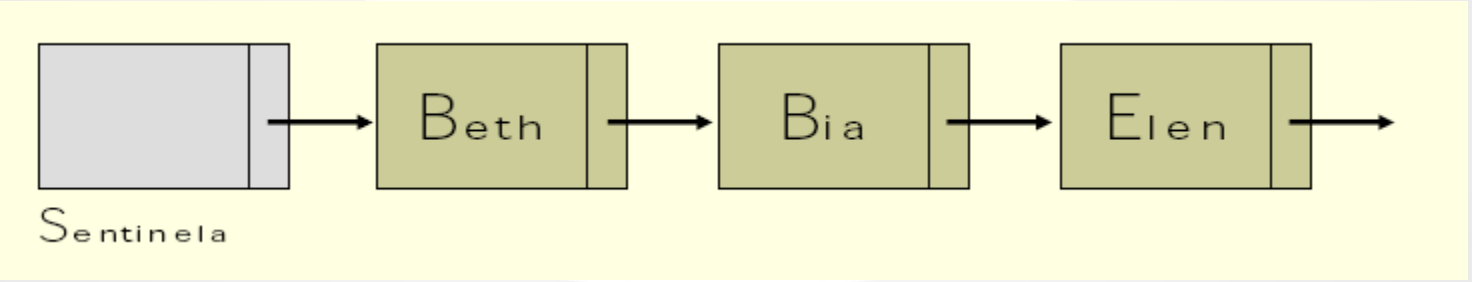
Lista Ligada com nó cabeça

Introdução

- Das operações com listas, a mais complexa é a remoção de um elemento do meio da lista
- Isso por que o algoritmo precisa apontar para o item anterior ao que será removido, o que no caso da remoção do primeiro elemento se configura como uma exceção que precisa ser tratada a parte



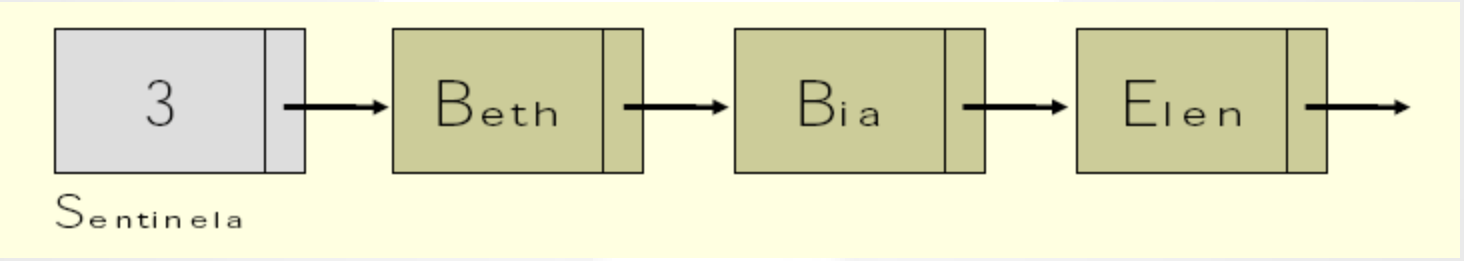
Introdução

- Das operações com listas, a mais complexa é a remoção de um elemento do meio da lista
- Isso por que o algoritmo precisa apontar para o item anterior ao que será removido, o que no caso da remoção do primeiro elemento se configura como uma exceção que precisa ser tratada a parte

- Uma solução que simplifica a implementação é substituir o ponteiro para o inicio por um nó cabeça
- Um nó cabeça é um nó normal da lista, mas esse é sempre o primeiro nó e a informação armazenada não tem valor

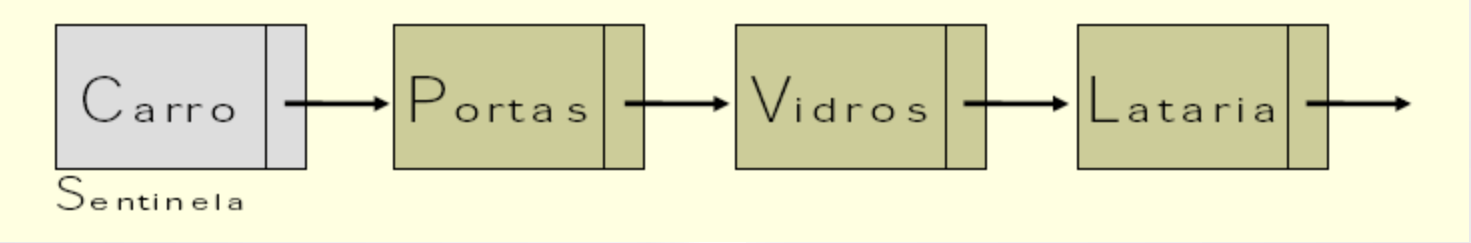
Introdução

- Possibilidades de uso
  - Informação global sobre a lista que possa ser necessária na aplicação
    - Armazenar número de elementos da lista, para que não seja necessário atravessá-la contando seus elementos



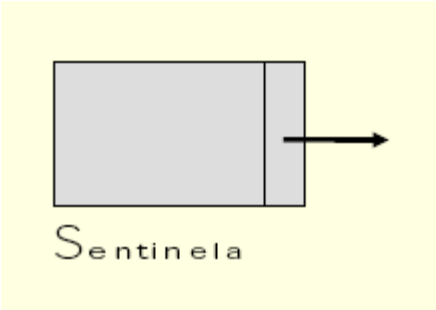
Introdução

- Possibilidades de uso
  - Informação global sobre a lista que possa ser necessária na aplicação
    - Em uma fábrica, guarda-se as peçasque compõem cada equipamento produzido, sendo este indicado pelo nó sentinela
    - Informações do vôo correspondente a uma fila de passageiros



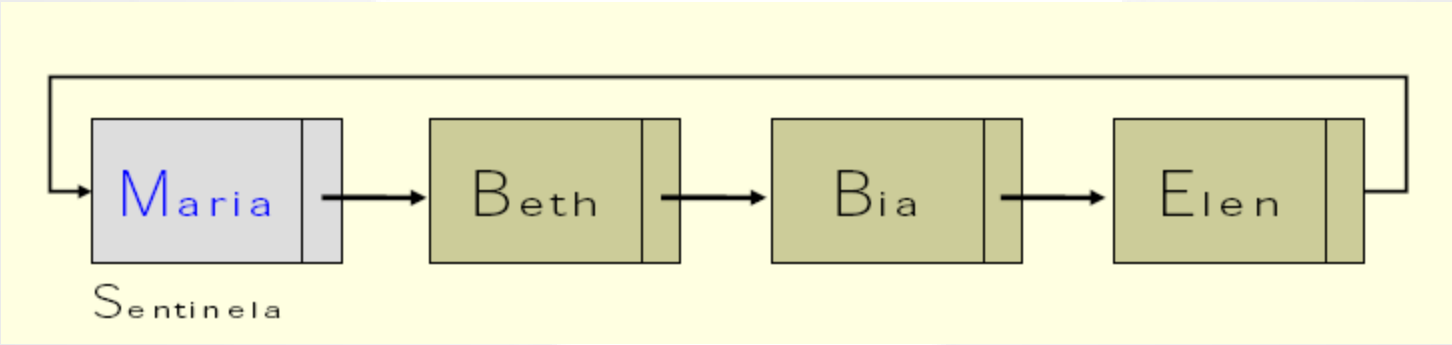
Introdução

- Possibilidades de uso
  - Informação global sobre a lista que possa ser necessária na aplicação
    - Lista vazia contém somente o nó sentinela



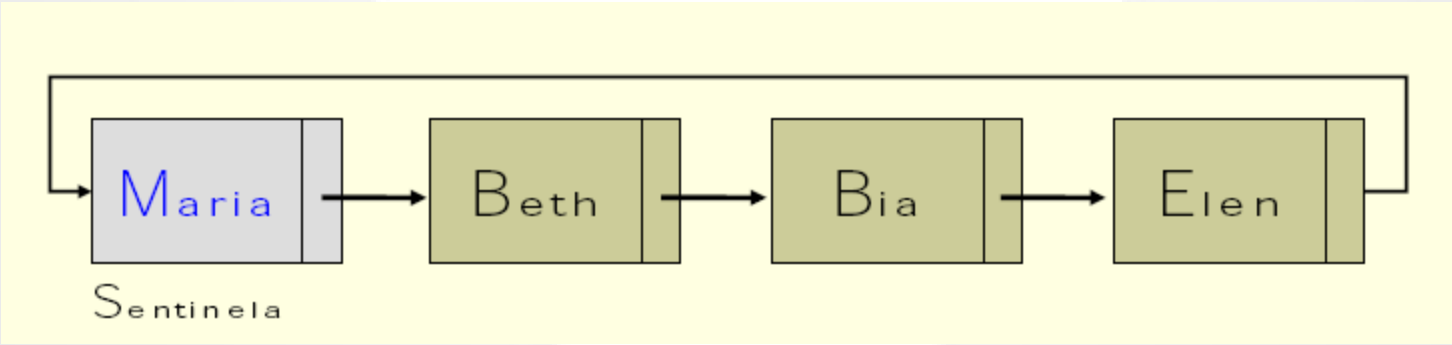
Introdução

- Possibilidades de uso
  - Lista circular
    - Uso da sentinela para simplificar Busca
      - Sempre vai encontrar a chave: se sentinela, então chave não estava na lista.



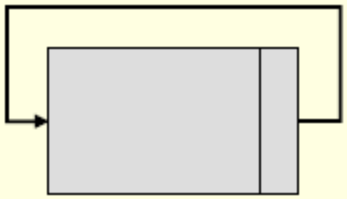
Introdução

- Possibilidades de uso
  - Lista circular
    - Como saber qual é o último elemento da lista?
      - Ele aponta para o nó sentinela



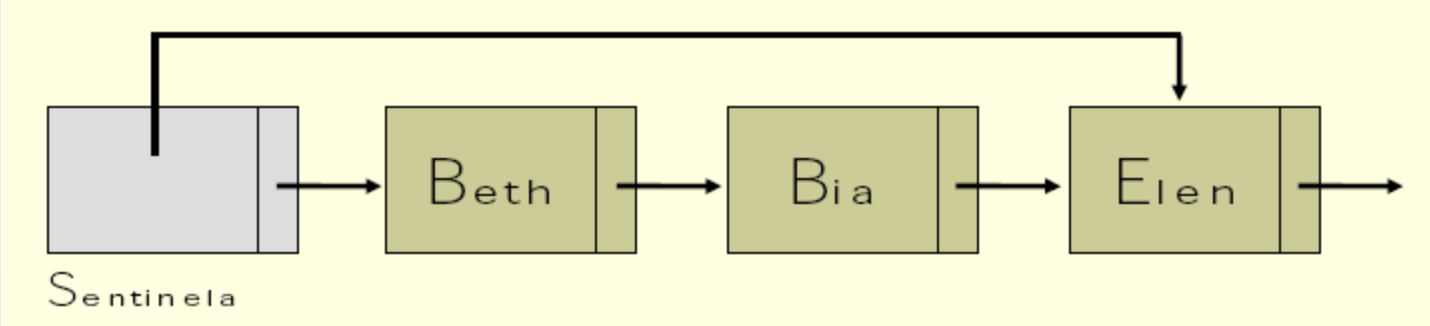
Introdução

- Possibilidades de uso
  - Lista circular
    - Como representar a lista vazia?



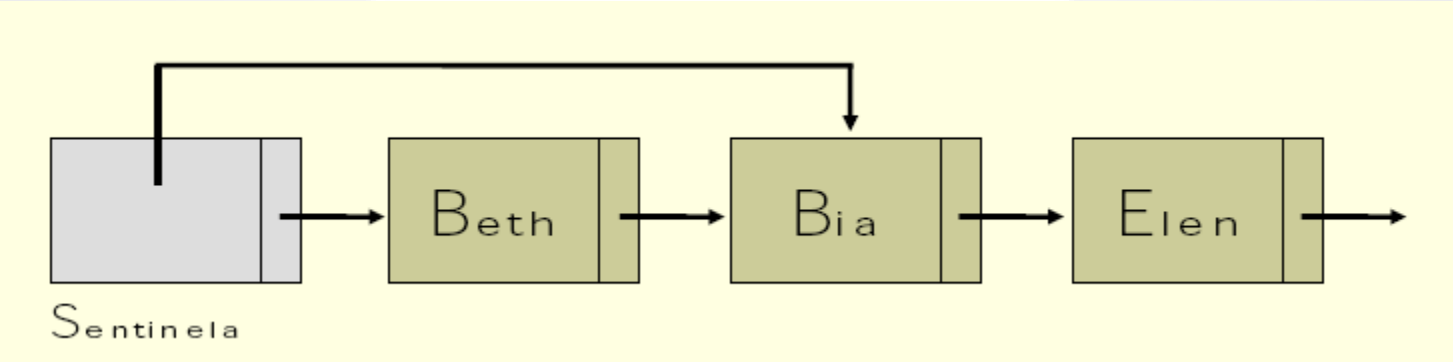
Introdução

- Possibilidades de uso
  - Informações para uso da lista como pilha, fila, etc.
    - Exemplo: em vez de um ponteiro de fim da fila, o nó sentinela pode apontar o fim
    - O campo info do nó sentinela passa a ser um ponteiro
    - Acaba por indicar o início da fila também



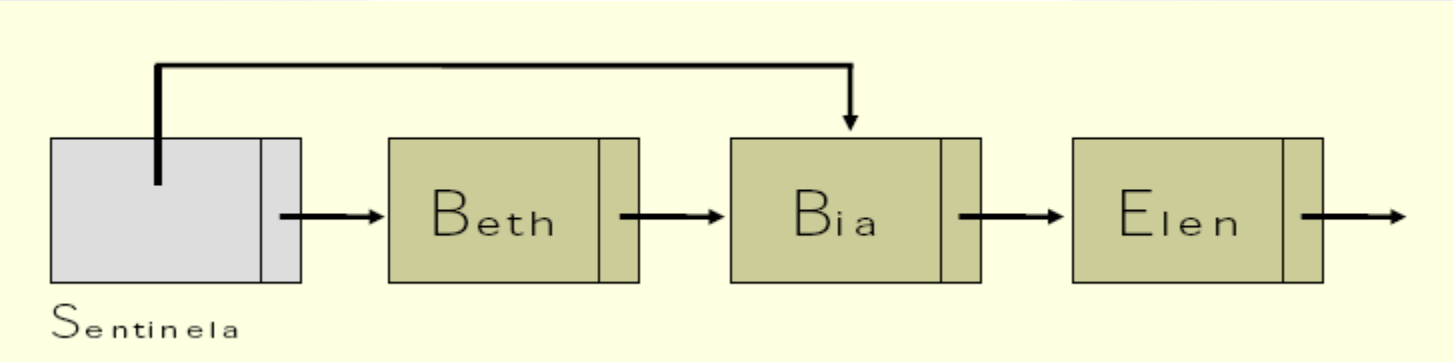
Introdução

- Possibilidades de uso
  - Indica um nó específico da lista
  - Por exemplo, em buscas que são constantemente interrompidas
    - Verificação de pessoas em ordem alfabética: poupa o esforço de se recomeçar ou a necessidade de ter uma variável auxiliar



Introdução

- Possibilidades de uso
  - Nó sentinela com ponteiro em seu campo info
    - Vantagem:
      - acesso possivelmente mais direto e imediato
    - Desvantagens?Quais?
      - Registro sentinela tem tipo distinto dos demais



Lista com nó cabeça

Sem nó cabeça

Acelerar a operação de inserção

Com nó cabeça

```
#define MAX 10

typedef struct {
  int chave;
  int valor;
} ITEM;
//representa a lista de itens

typedef struct NO {
    ITEM item;
    struct NO *proximo;
} tNO;
```

```
#define MAX 10

typedef struct {
  int chave;
  int valor;
} ITEM;
//representa a lista de itens

typedef struct {
    ITEM item;
    struct NO *proximo;
} tNO;
```

```
typedef struct {
   tNO *inicio;
   TNO *fim;
} LISTA_Ligada;
```

```
typedef struct {
    tNO cabeca;
    tNO *fim;
} LISTA_Ligada_Cabeca;
```

Única alteração

Poderia ser um ponteiro

Nó cabeça e Lista vazia

- A lista com nó cabeça será vazia quando o próximo do nó cabeça apontar para NULL

Sem nó cabeça

```
int vazia(Lista_ligada_cabeca *lista){

 return (lista->inicio ==NULL);
}
```

Com nó cabeça

```
int vazia(Lista_ligada_cabeca *lista){

  return (lista->cabeca.proximo ==NULL)
}
```

Implementação das demais operações

- A implementação das demais operações é similar a lista ligada padrão (sem nó cabeça), a única alteração é substituir as referências ao ponteiro início pelo próximo nó cabeça
- O grande ganho é na remoção do meio da lista (por posição), já que nesse não é necessário tratar separadamente quando o item a remover é o primeiro.

Lista Ligadas com cabeça

Sem nó cabeça

```
void criar (Lista_Ligada *lista){
    lista->inicio = NULL;
}
```

Ponteiros inicio

```
void criar (Lista_Ligada *lista){
    lista->inicio = NULL;
    lista->fim = NULL;
}
```

Ponteiros inicio, fim

Com nó cabeça

```
void criar (Lista_Ligada_Cabeca *lista){
    lista->fim = &lista->cabeca;
    lista->cabeca.prox=NULL;
}
```

Lista Ligadas com cabeça

Sem nó cabeça

Ponteiros inicio, fim

```
void apagar_lista (Lista_Ligada *lista){
    if (!vazia(lista)){
        tNO *paux = lista->inicio;
        while (paux != NULL) {
          tNO *prem = paux;
          paux = paux->proximo;
          free(prem);
        }
    }
    lista->inicio=NULL;
    lista->fim = NULL;
}
```

Com nó cabeça

```
void apagar_lista (Lista_Ligada *lista){
    if (!vazia(lista)){
        tNO *paux = lista->cabeca.prox;
        while (paux != NULL) {
          tNO *prem = paux;
          paux = paux->proximo;
          free(prem);
        }
    }
    lista->cabeca.prox = NULL;
    lista->fim = &lista->cabeca;
}
```

Sem nó cabeça

Ponteiros inicio, fim

```
int inserir_posicao (Lista_Ligada *lista, int pos, ITEM *item){

   tNO *pnovo = (tNO *)malloc (sizeof(tNO)); //cria um novo nó
   if (pnovo != NULL) {//verifica se existe memoria disponivel
      pnovo->item = *item;
      if (pos==0) {//adiciona na primeira posicao
         pnovo->proximo = lista->inicio;
         lista->inicio = pnovo;
       } else {
       tNO *patual = lista->inicio;
       tNO *pant ;
       //encontra a posição de inserção
       for (int i=0; i<pos; i++){
           if (patual != lista->fim) {
               pant = paux;
               patual = patual->proximo;
           } else{
           return 0;
           }
        }
        // faz as ligações para a inserção do novo elemento
        pnovo->proximo =patual;
        pant->proximo = pnovo;
     }
     return 1
    }
    else {
      return 0;}
}
```

Ponteiro para o fim não é alterado

Com nó cabeça

Ponteiro fim

```
int inserir_posicao (Lista_Ligada_Cabeca *lista, int pos, ITEM *item){

   tNO *pnovo = (tNO *)malloc (sizeof(tNO)); //cria um novo nó
   if (pnovo != NULL) {//verifica se existe memoria disponivel
       pnovo->item = *item;
       tNO *pant = &lista->cabeca;
       tNO *patual = pant->prox;
       //encontra a posição de inserção
       for (int i=0; i<pos; i++){
           if (patual != lista->fim) {
               pant = paux;
               patual = patual->proximo;
           } else{
           return 0;
           }
        }
        // faz as ligações para a inserção do novo elemento
        pnovo->proximo =patual;
        pant->proximo = pnovo;
     }
     return 1
    }
    else {
      return 0;}
}
```

```
int remover_posicao (Lista_Ligada *lista, int pos){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
       int i;
       tNO *pant ;
       tNO *patual = lista->inicio;
       //encontra a posição de remoção
       for (i=0;i<pos;i++) {
            if (paux != lista->fim) {
                pant = paux;
                paux = paux->proximo;
            }else { //posição fora da lista
                return 0;
            }
         }
         if (paux == lista->inicio ) {//remove o primeiro item
             lista->inicio = paux->proximo;
         } else if (paux == lista->fim) {//remove o ultimo item
            lista->fim = pant;
            pant->proximo = NULL;
         }else {//remove item no meio da lista
            pant->proximo = paux->proximo;
         }
        free(paux); //remove o item da memoria
        return 1;
      }
    return 0;
  }
```

Remover um item

Com nó cabeça

Ponteiro fim

```
int remover_posicao(Lista_Ligada_cabeca *lista, int pos){
    int i;
    if (!vazia(lista)){//verifico se a lista está vazia
       //aponta para o elemento anterior a ser retirado
       NO *paux = &lista->cabeca;
       for (i = 0; i < pos; i++){
          if (paux->proximo!=lista->fim) {
              paux = paux->proximo;
          }else{
              return 0;
          }
       }
       NO *prem = paux->proximo;
       paux->proximo = paux->proximo->proximo;

       if (prem->proximo == NULL) {//retirei o ultimo item
          lista->fim = paux;
       }
       free(prem);
       return 1;
     }else{
       return 0;
   }
}
```

Lista Ligadas com cabeça

Sem nó cabeça

Ponteiros inicio, fim

```
void exibir (Lista_Ligada *lista){
   if (!vazia(lista)){
      tNO *paux = lista->inicio;
      while (paux != NULL) {
         printf('Chave =%d, Valor%d',paux-> item. chave,
                                        paux->item.valor)
         paux = paux->proximo;
      }
   }else{
      printf('Lista Vazia');
   }
}
```

Com nó cabeça

```
void exibir (Lista_Ligada *lista){
   if (!vazia(lista)){
      tNO *paux = lista->cabeca.prox;
      while (paux != NULL) {
        printf('Chave =%d, Valor%d',paux-> item. chave,
                                        paux->item.valor)
      }
  }else{
     printf('Lista Vazia');
  }
}
```

Lista Ligadas com cabeça

Sem nó cabeça

Ponteiros inicio, fim

```
int buscaseq (Lista_Ligada *lista, int chave, ITEM *item)
  {
       tNO *paux = lista->inicio;
       while (paux != NULL) {
          if (paux->item.chave == chave) {
             *item = paux->item;
              return 1;
           }
           paux = paux->proximo;
        }
        return 0;
   }
```

Com nó cabeça

```
int buscaseq (Lista_Ligada *lista, int chave, ITEM *item)
  {
       tNO *paux = lista->cabeca.prox;
       while (paux != NULL) {
          if (paux->item.chave == chave) {
             *item = paux->item;
              return 1;
           }
           paux = paux->proximo;
        }
        return 0;
   }
```

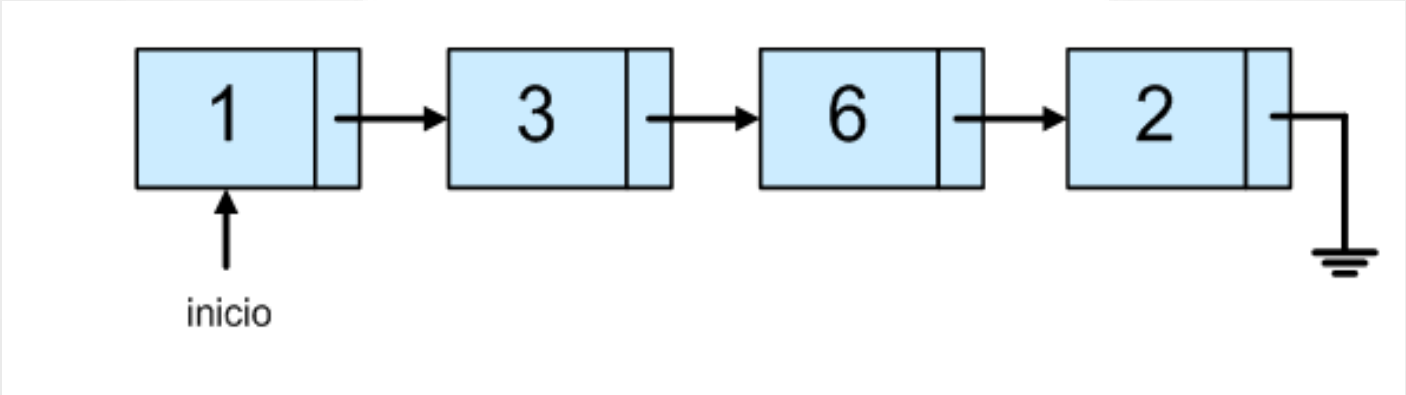
Exercícios

- Implementar as demais operações do TAD listas usando o conceito de lista ligada com nó cabeça

Lista Ligada Circular com Sentinela

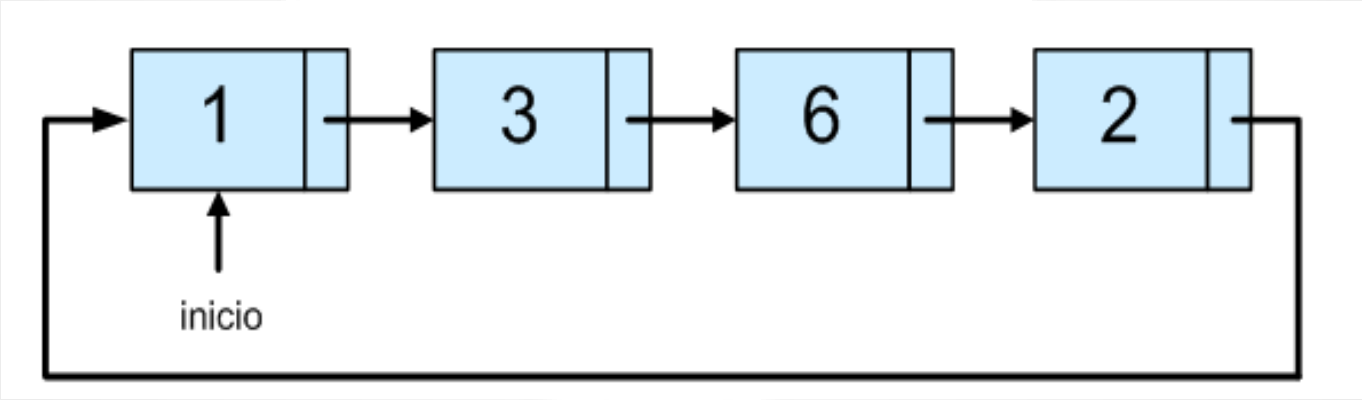
Lista Ligada

- Um diferente tipo de implementação de listas ligadas substitui a definição de que o próximo do último é NULL por o próximo do último é o primeiro.



Lista Ligada Circular

- Um diferente tipo de implementação de listas ligadas substitui a definição de que o próximo do último é NULL por o próximo do último é o primeiro.

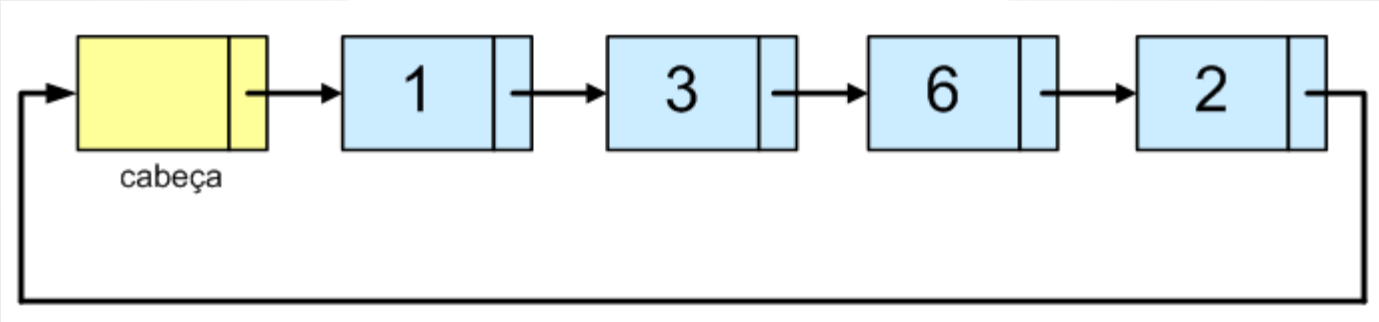


Lista Ligada Circular

- A partir de um nó da lista pode-se chegar a qualquer outro nó
- Nessa implementação somente um ponteiro para o fim da lista é necessário, não sendo necessário um ponteiro para o início. Isso por que o início é o próximo do fim.

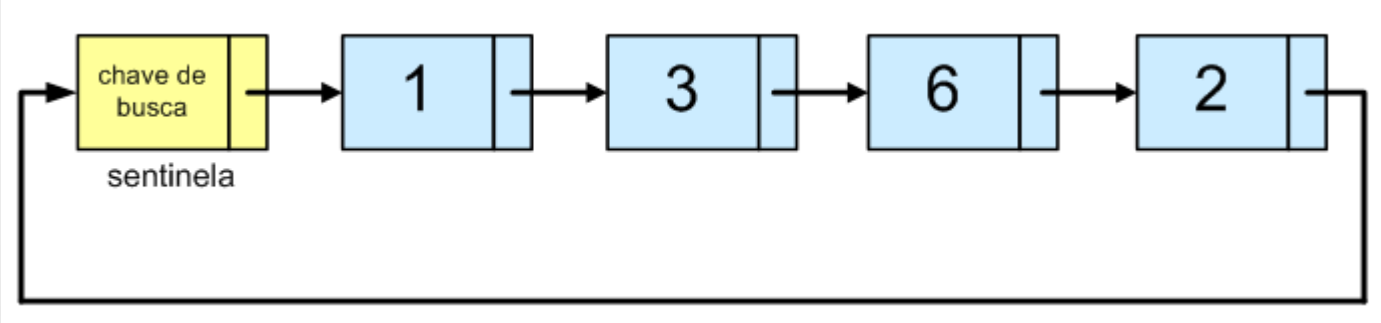
Listas Ligadas Circulares (Sentinela)

- No caso especial da busca em listas circulares, o emprego de um nó cabeça pode reduzir a quantidade de testes necessários
- A ideia é colocar a chave de busca no nó cabeça e começar a busca no próximo nó
- Se o item encontrado for a cabeça, a busca não teve sucesso. Assim um teste é "economizado" já que não é preciso testar se a lista acabou.
- Nesse caso, o nó cabeça é chamado de sentinela.



Listas Ligadas Circulares (Sentinela)

- No caso especial da busca em listas circulares, o emprego de um nó cabeça pode reduzir a quantidade de testes necessários
- A ideia é colocar a chave de busca no nó cabeça e começar a busca no próximo nó
- Se o item encontrado for a cabeça, a busca não teve sucesso. Assim um teste é "economizado" já que não é preciso testar se a lista acabou.
- Nesse caso, o nó cabeça é chamado de sentinela.



Listas Ligadas Circulares (Sentinela)

Nó cabeça

Não circular

Circular

```
#define MAX 10

typedef struct {
  int chave;
  int valor;
} ITEM;
//representa a lista de itens

typedef struct {
    ITEM item;
    struct NO *proximo;
} tlista;
```

```
#define MAX 10

typedef struct {
  int chave;
  int valor;
} ITEM;
//representa a lista de itens

typedef struct {
    ITEM item;
    struct NO *proximo;
} tlista;
```

```
typedef struct {
    tNO cabeca;
    tNO *fim;
} LISTA_Ligada;
```

```
typedef struct {
    tNO sentinela;
    tNO *fim;
} LISTA_Circular_Cabeca;
```

Única alteração

Busca em Lista Circular com Sentinela

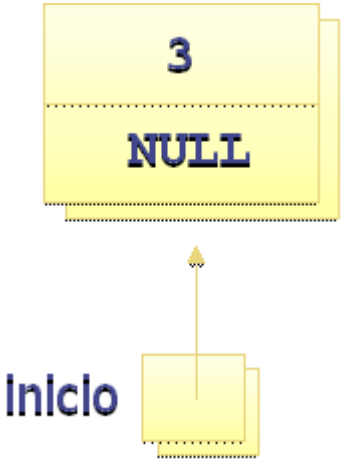
```
int buscar(Lista_Circular_Cabeca *lista, int chave, ITEM *item){
   //atribui a chave ao sentinela
    lista->sentinela.item.chave = chave;

     NO *paux = &lista->sentinela;

    do {
       paux = paux->proximo;
    } while (paux->item.chave != chave)

     *item = paux->item; //retorno o valor encontrado

     return (paux !=&lista->sentinela);
    // verifico se o valor não é a sentinela
}
```

Tipos Listas

Lista Ligada Ordenada

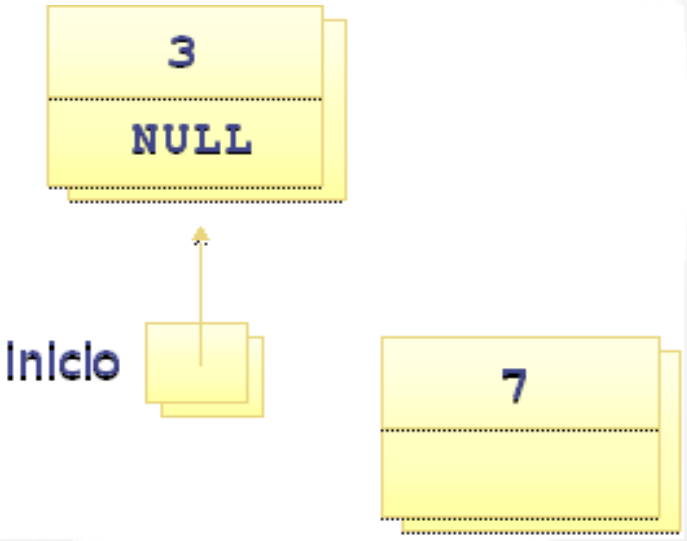
Inserção Ordenada – Lista Ligada

- Inserindo valor 3



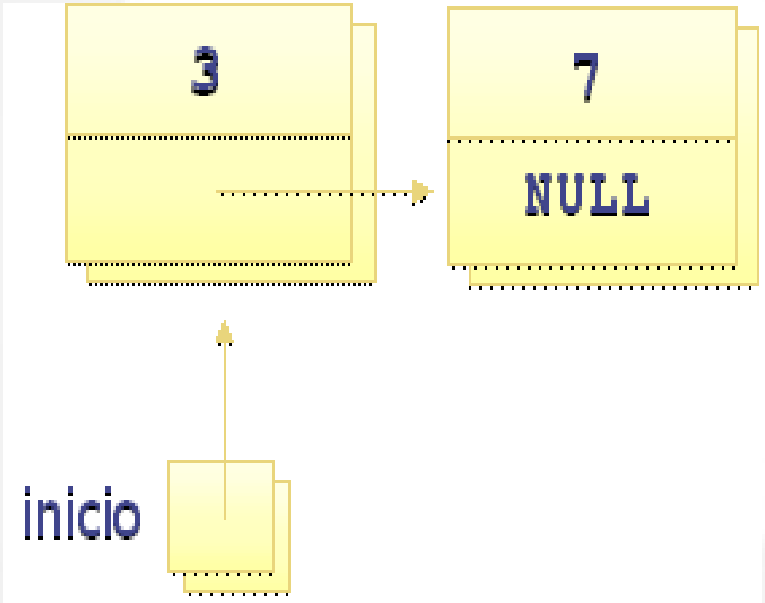
Inserção Ordenada – Lista Ligada

- Inserindo valor 7



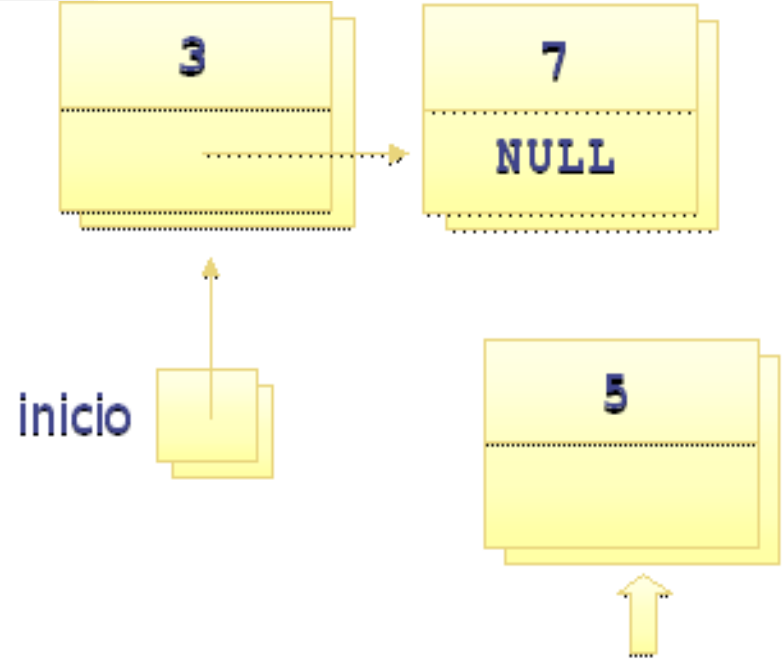
Inserção Ordenada – Lista Ligada

- Inserindo valor 7



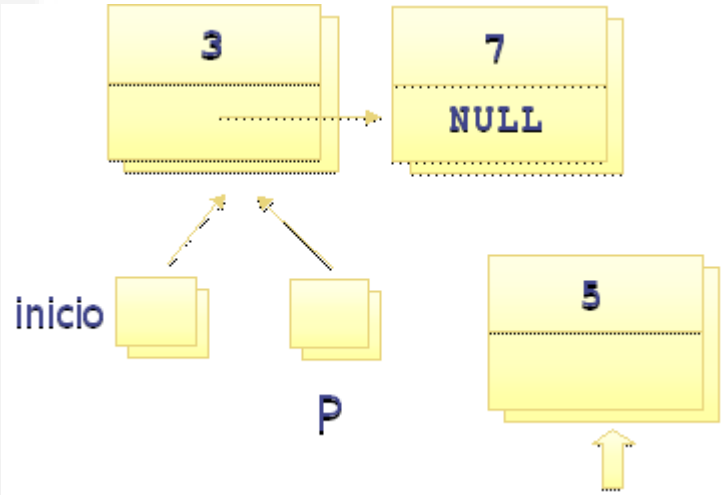
Inserção Ordenada – Lista Ligada

- Inserindo valor 5



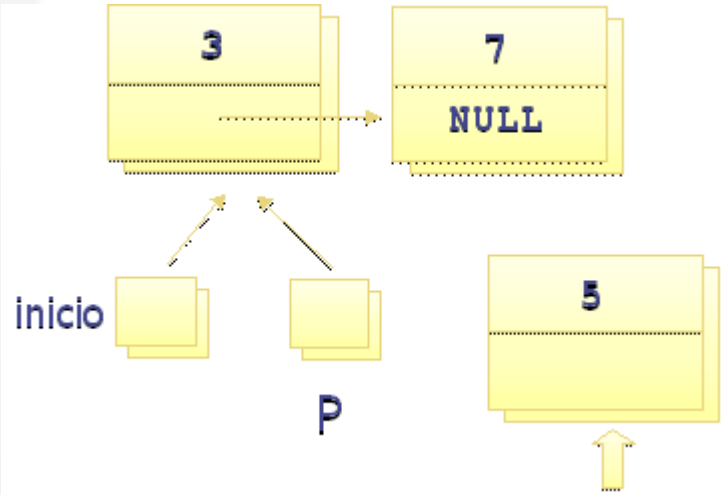
Inserção Ordenada – Lista Ligada

- Inserindo valor 5
- inicio.chave menor que novo.chave



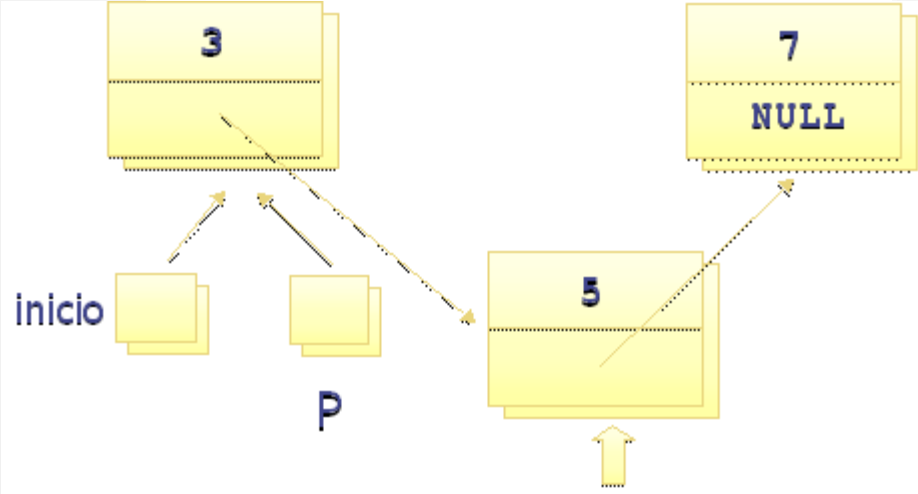
Inserção Ordenada – Lista Ligada

- Inserindo valor 5
- p->proximo.chave maior que novo.chave



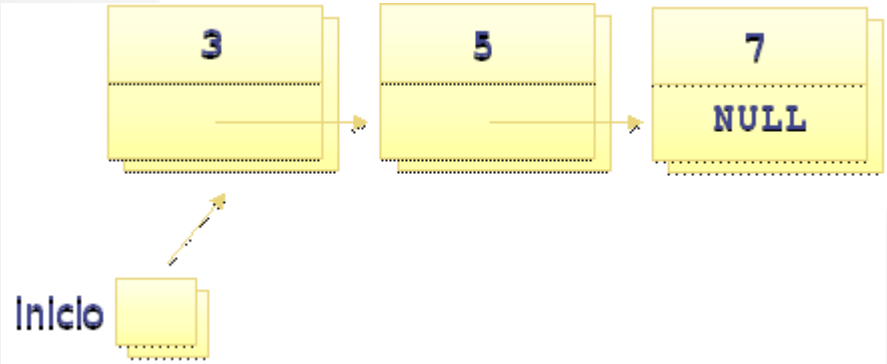
Inserção Ordenada – Lista Ligada

- Inserindo valor 5



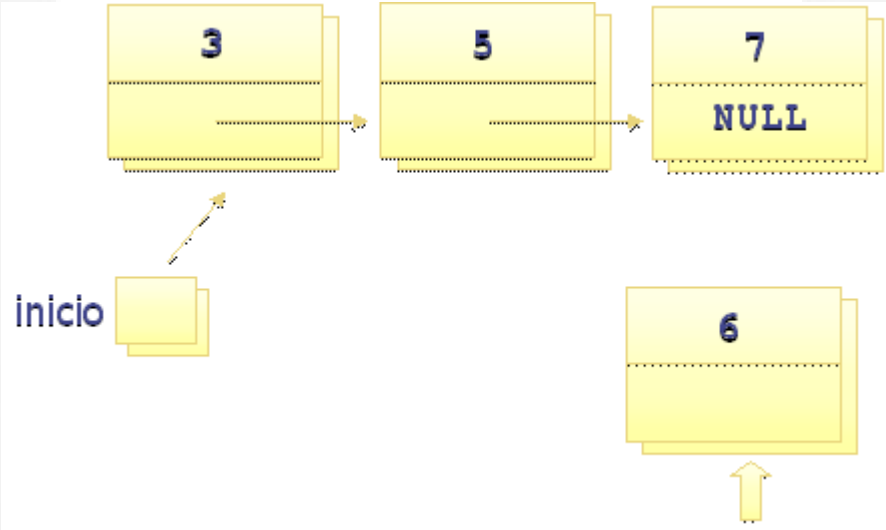
Inserção Ordenada – Lista Ligada

- Inserindo valor 5



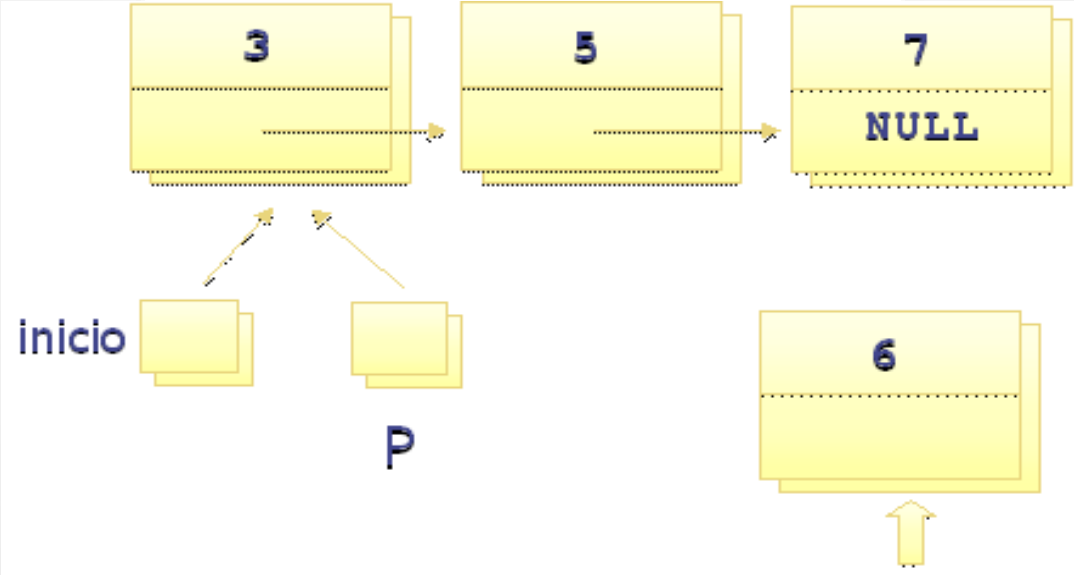
Inserção Ordenada – Lista Ligada

- Inserindo valor 6
- inicio.chave menor que novo.chave



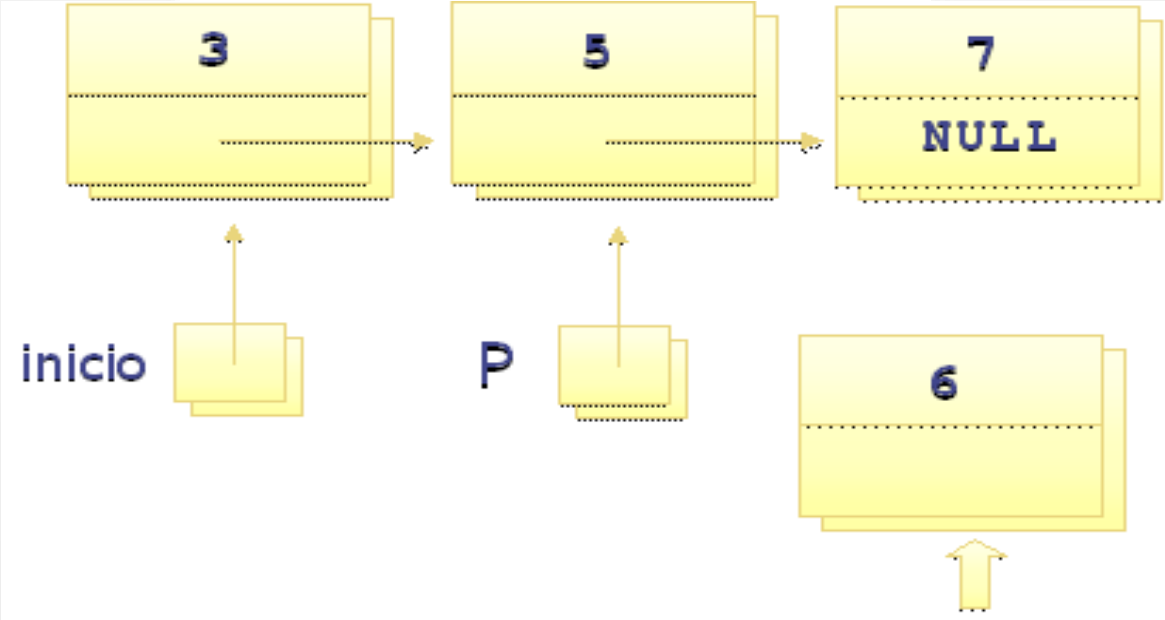
Inserção Ordenada – Lista Ligada

- Inserindo valor 6
- p->proximo.chave menor que novo.chave



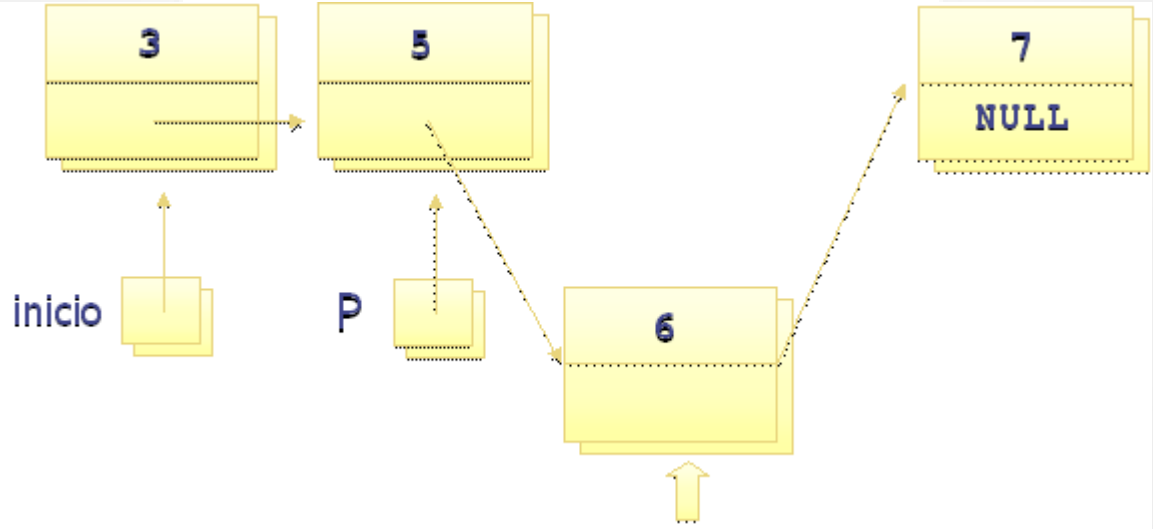
Inserção Ordenada – Lista Ligada

- Inserindo valor 6
- p->proximo.chave maior que novo.chave



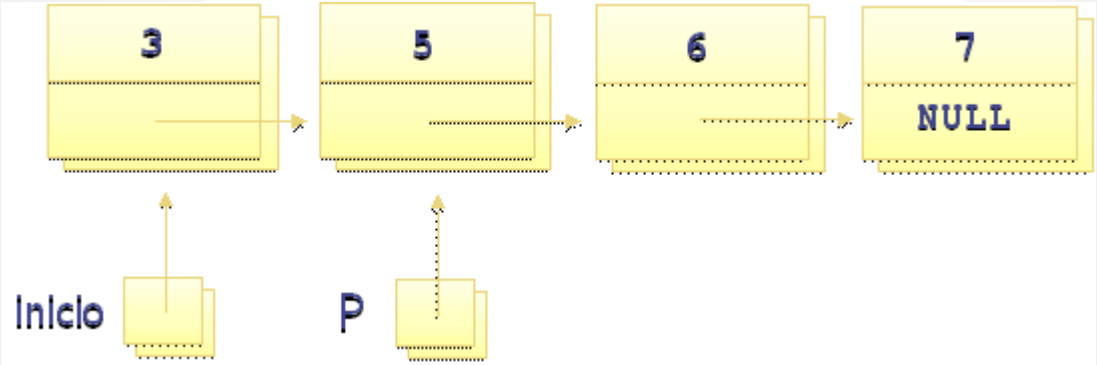
Inserção Ordenada – Lista Ligada

- Inserindo valor 6
- p->proximo.chave maior que novo.chave



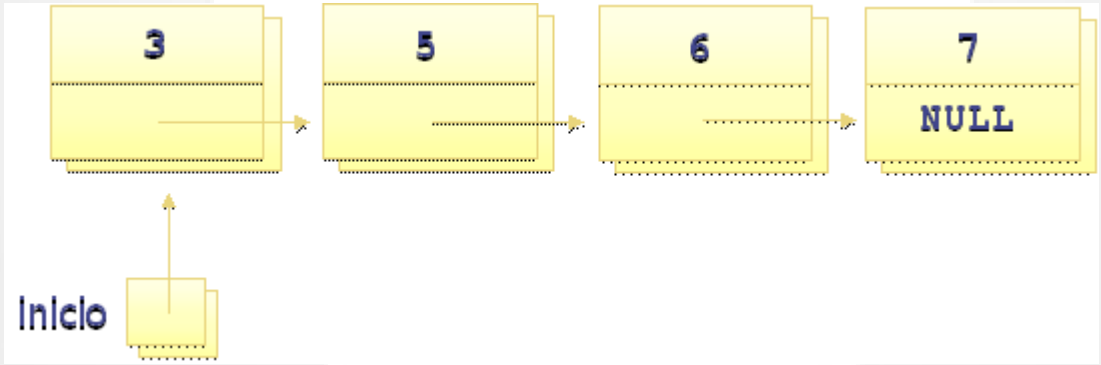
Inserção Ordenada – Lista Ligada

- Inserindo valor 6



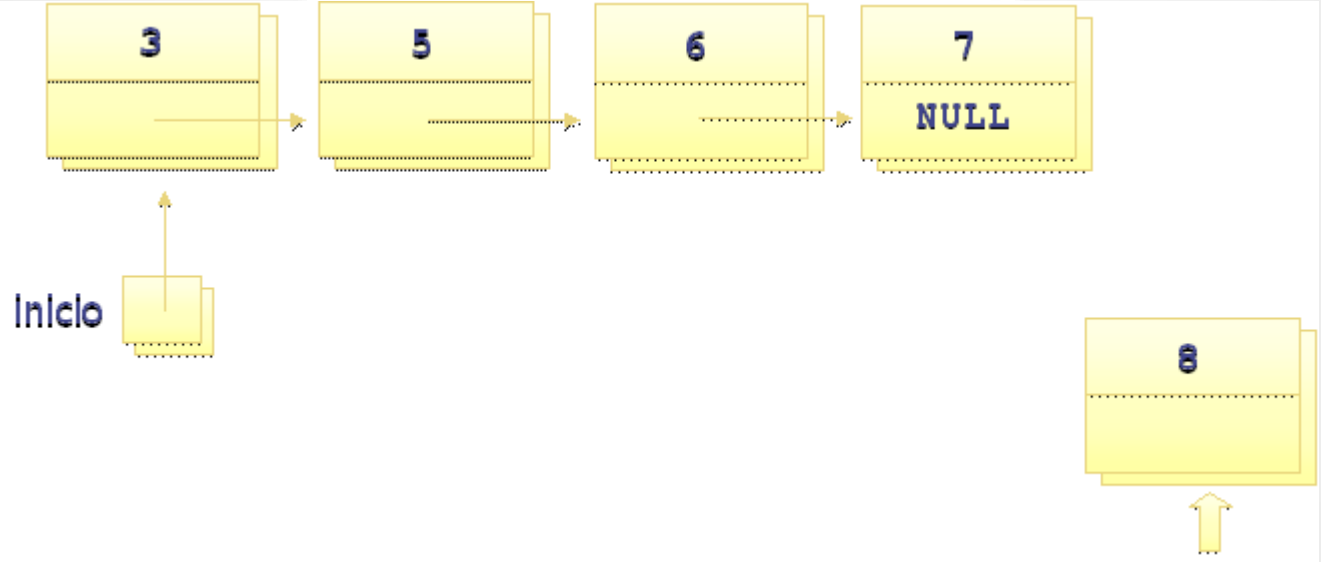
Inserção Ordenada – Lista Ligada

- Inserindo valor 6



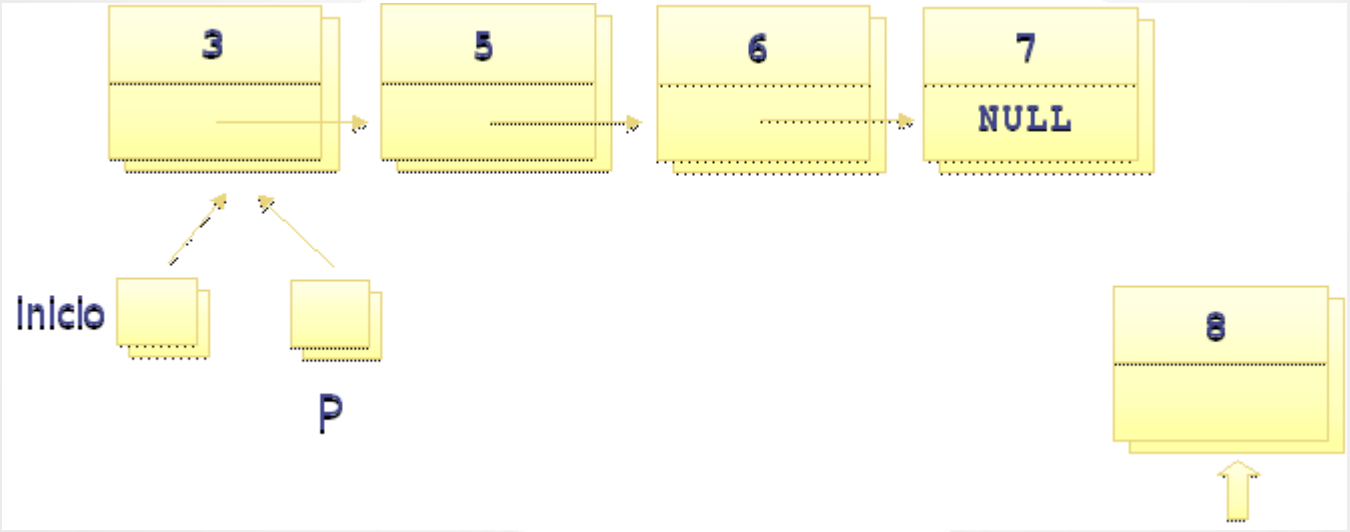
Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- inicio.chave menor que novo.chave



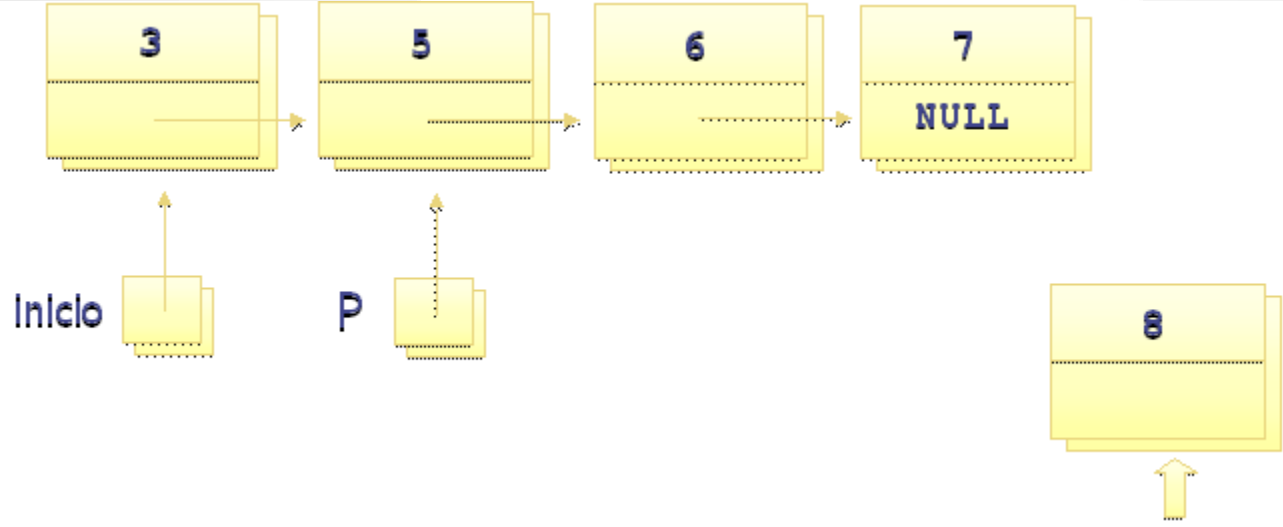
Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- p->proximo.chave menor que novo.chave



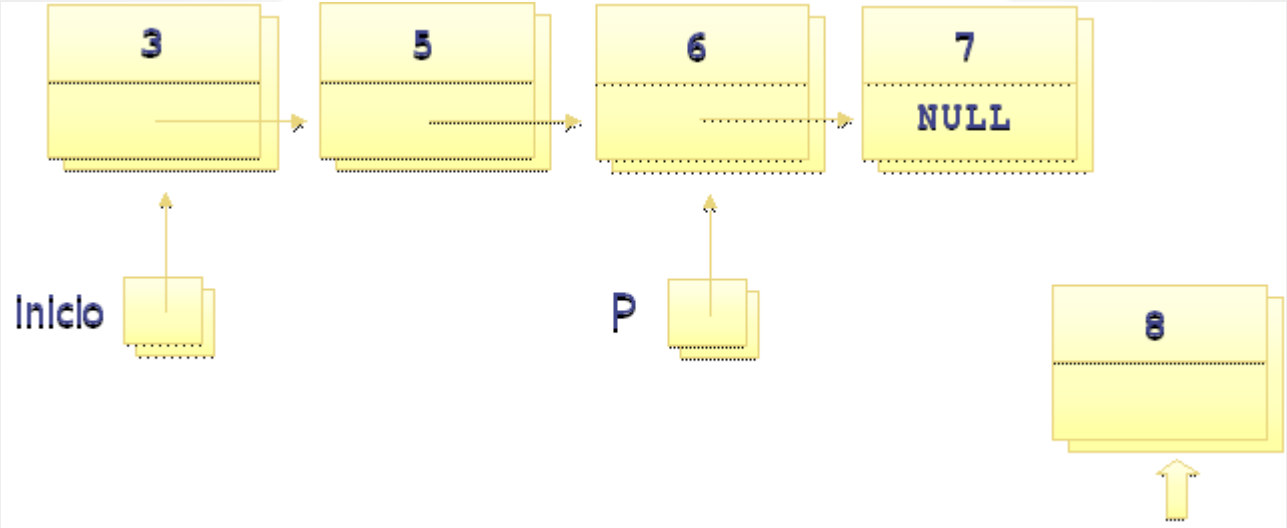
Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- p->proximo.chave menor que novo.chave



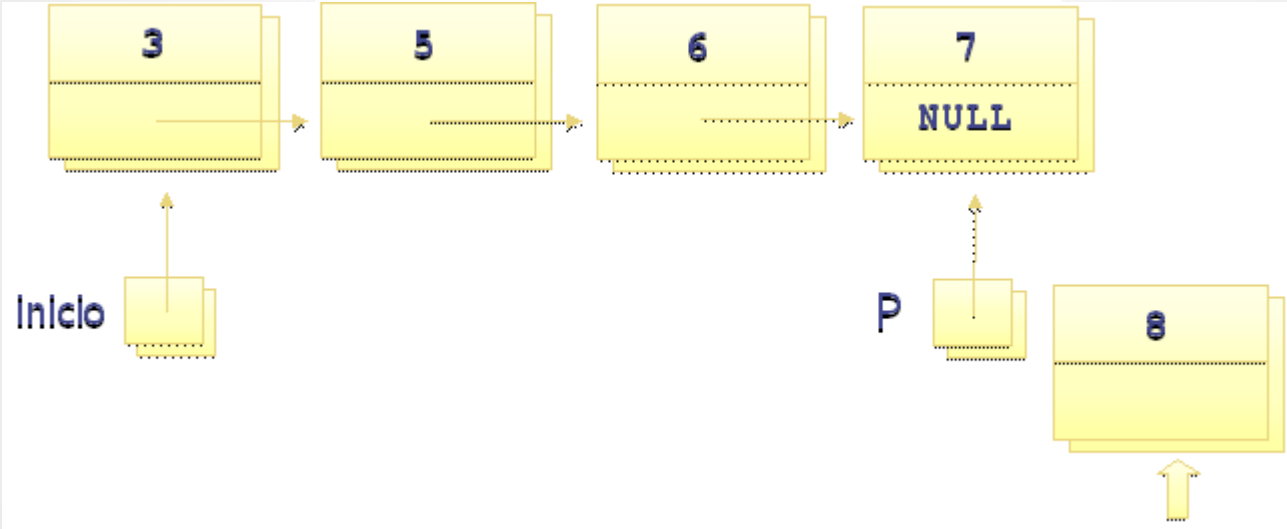
Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- p->proximo.chave menor que novo.chave



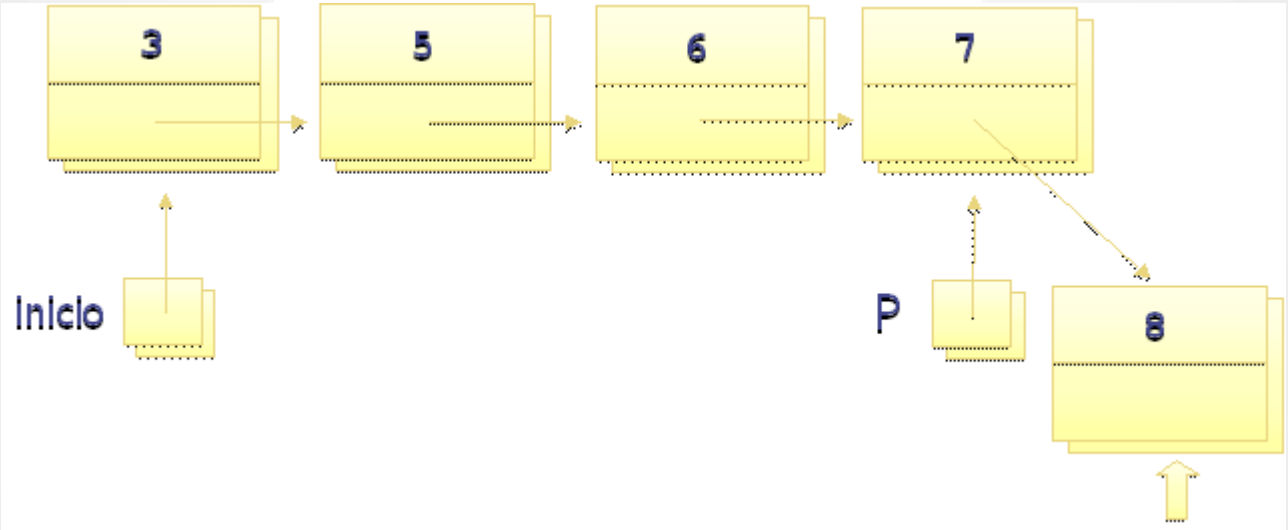
Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- p->proximo.chave não existe



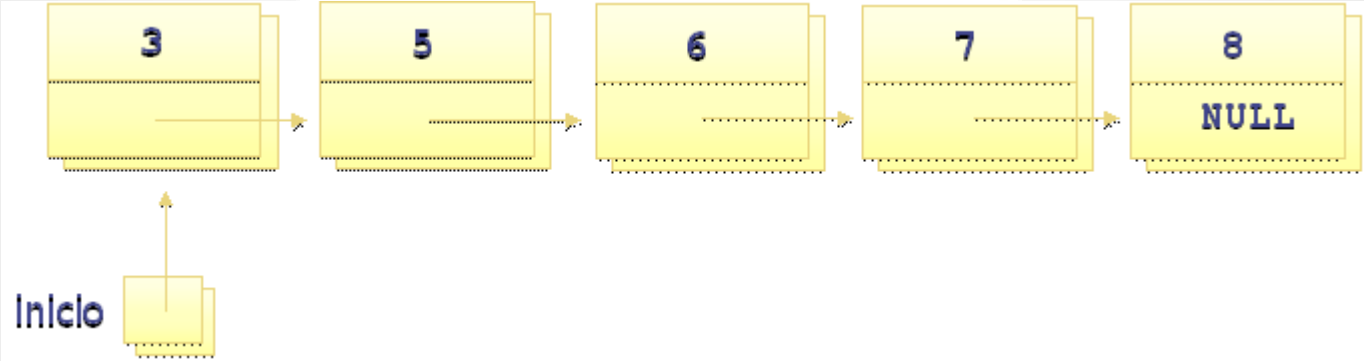
Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- p->proximo.chave não existe



Inserção Ordenada – Lista Ligada

- Inserindo valor 8
- p->proximo.chave não existe



Comentários sobre a implementação

- Não precisa do ponteiro fim porque a inserção será em qualquer posição de lista
- Novamente o emprego do nó cabeça facilita a implementação uma vez que vamos buscar a posição anterior da inserção, e no caso de ser o menor item da lista isso não representará exceção

Inserção em Lista Ligada com cabeça Ordenada

```
Int inserir(Lista_Ligada_Cabeca *lista,  ITEM *item){

    //cria um novo nó
   tNO *pnovo = (tNO *) malloc(sizeof(NO));
   if (pnovo != NULL) {
     pnovo->item = *item;//preencho com os dados
     pnovo->proximo = NULL; //defina que o próximo é nulo

     //armazena posição anterior da inserção
     tNO *paux = &lista->cabeca;

     while ((paux->proximo!=NULL)&&
            (paux->proximo->item.chave<item->chave)){

           paux = paux ->proximo;
     }
     pnovo ->proximo =paux->proximo;
     paux->proximo = pnovo;
     return 1;
   }else{
     return 0;
  }
}
```

Busca em Lista Ordenada

- Lembrete: é possível tirar vantagem em uma busca se a lista é ordenada

```
int buscar(Lista_ligada_Cabeca *lista,int chave,ITEM *item)
{

    //cria um novo nó
   tNO *pnovo = lista->cabeca.proximo;

   while (paux !=NULL){
      if (paux->item.chave == chave) {
            //se a chave for igual
            *item = paux ->item;
            return 1;
      }else if (paux->item.chave > chave)
            //se a chave na lista for maior
            return 0;
     }
     paux = paux->proximo;
   }
   return 0;
}
```

Lista Ordenada – Outras Operações

- As demais operações implementadas podem deixar a lista desordenada?
- Poderia ocorrer com a remoção de elementos, entretanto
  - Com vetores, a implementação deslocava os elementos
  - Com listas ligadas, os nós removidos não alteram a ordem dos demais