Fila Estática Circular

Relembrando

- Filas são estruturas lineares nas quais as inserções são feitas em um extremo (final) e remoções são feitas no outro extremo (início)
- Seguem a política “First-in, First-out” (FIFO)
- Modelos intuitivos de filas são as linhas para comprar bilhetes de cinema e de caixa de supermercado

- Exemplos de aplicações de filas
  - Filas de espera e algoritmos de simulação
  - Controle por parte do sistema operacional a recursos compartilhados, tais como impressoras
  - Buffers de Entrada/Saída
  - Estrutura de dados auxiliar em alguns algoritmos como a busca em largura

Relembrando : Operações Principais

- inserir(F,x): insere o elemento x no final da fila F e retorna true se foi possível inserir false caso contrário
- remover (F,x): remove o elemento no inicio de F, e retorna esse elemento x. Retorna true se foi possível remove-lo

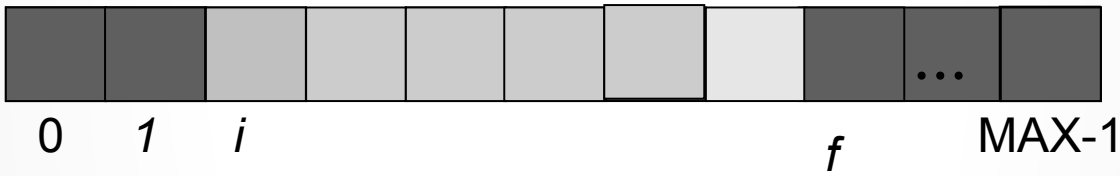
Relembrando : Implementação estática

- Uma maneira simples de implementar uma fila é fixar o início da fila na primeira posição do vetor
- As inserções (enfileirar) fazem com que o contador n seja incrementado (O(1))
- Mas as remoções (desenfileira) requerem deslocar todos os elementos (O(n))


```
#define MAX 10

typedef struct {
    int valor;
} ITEM;

typedef int ITEM;

typedef struct {
      ITEM  item[MAX];
      int n;
 } fila;
```

Implementação Estática Circular

- A solução é fazer com que o início não seja fixo na primeira posição do vetor
- Portanto, deve-se manter dois contadores para o início (i) e final da fila (f)



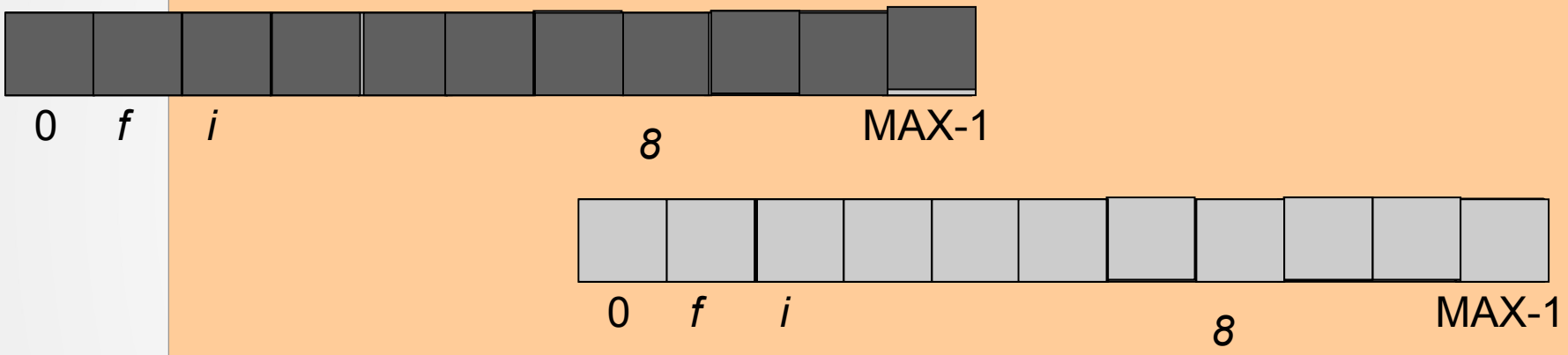
- Mas o que deve ser feito quando f = Max-1 e deseja-se inserir mais um elemento?

Implementação Estática Circular

- Pode-se permitir que f volte para o início do vetor quando esse contador atingir o final do vetor
- Essa implementação é conhecida como fila circular Na figura abaixo a fila circular possui 4 posições vagas e f < i

Implementação Estática Circular

- Caso o inicio aponte para o primeiro elemento inserido e o fim aponte para o último, não é possível distinguir se uma fila está cheia ou vazia



- Uma estratégia seria apontar o fim para a próxima posição de inserção
- **Assim, se a próxima posição do fim for igual ao inicio, a fila está cheia**
- Perde-se uma posição no vetor (a fila terá TAM-1 posições de inserção)

Definição de Tipos

Fila não circular

```
#define MAX 10

typedef struct {
      int valor;
} ITEM;

 typedef int ITEM;

typedef struct {
        ITEM  itens[MAX];
        int n;
 } fila;
```

Fila circular

```
#define MAX 10

typedef struct {
      int valor;
} ITEM;

 typedef struct {
        ITEM itens[TAM];
        int fim;
        int inicio;
} Fila_Circular
```

Implementação Estática Circular

```
void criar(FILA_ESTATICA *fila)
{
      fila->fim = 0;
      fila->inicio = 0;
}
```

```
int vazia(FILA_ESTATICA *fila) {
      return (fila->inicio == fila->fim);
}
```

```
int cheia(FILA_ESTATICA *fila) {
    return (((fila->fim+1) % MAX) ==  fila->inicio);
}
```

Implementação Estática Circular

```
int inserir(FILA_ESTATICA *fila, ITEM *item) {
  if (!cheia(fila)) {
   //insiro o item
   fila->itens[fila->fim] = *item;
   //avanço o fim para a próxima posição vazia
   fila->fim = (fila->fim+1) % MAX;
   return 1;
  }
 return 0;
}
```

```
int remove(FILA_ESTATICA *fila, ITEM *item) {
   if (!vazia(fila)) {
    //recupero o primeiro item
    *item = fila->itens[fila->inicio];
    //avanço o fim para próxima posição vazia
    fila->inicio = (fila->inicio+1) % MAX;
    return 1;
   }
  return 0;
}
```

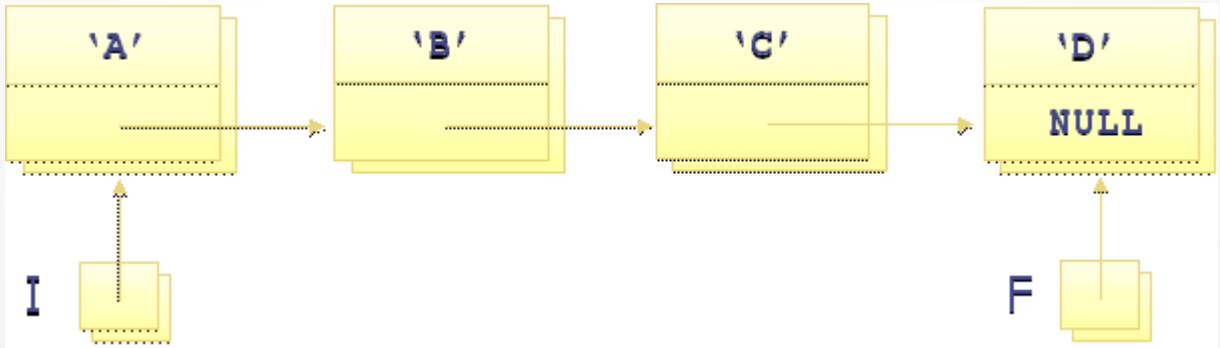
Implementação Estática Circular

```
int contar(FILA_ESTATICA *fila) {
    if (fila->fim >= fila->inicio) {
       return (fila->fim - fila->inicio);
    } else {
    return (MAX - (fila->inicio - fila->fim));
   }
}
```

Implementação Dinâmica

- A implementação dinâmica pode ser realizada mantendo-se dois ponteiros, um para o início e outro para o final da fila

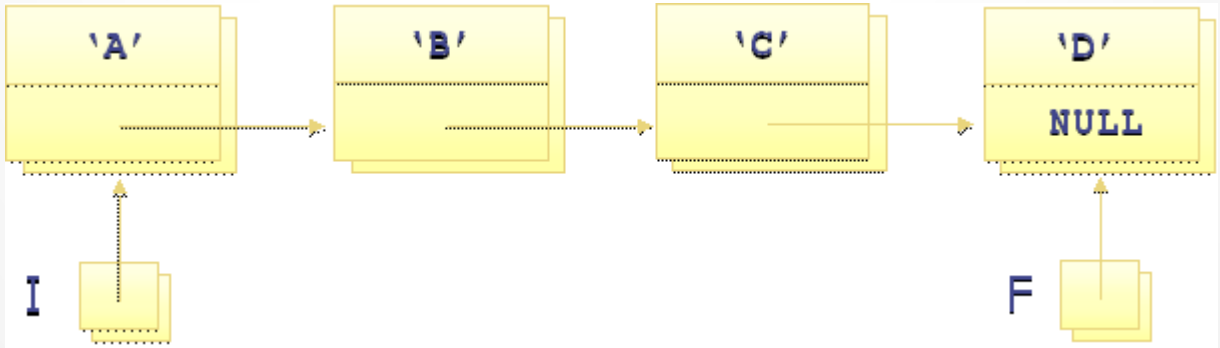
- Com isso pode-se ter acesso direto às posições de inserção e remoção



Implementação Dinâmica

- As operações inserir(enfileirar) e remover(desenfileirar) implementadas dinamicamente são bastante eficientes

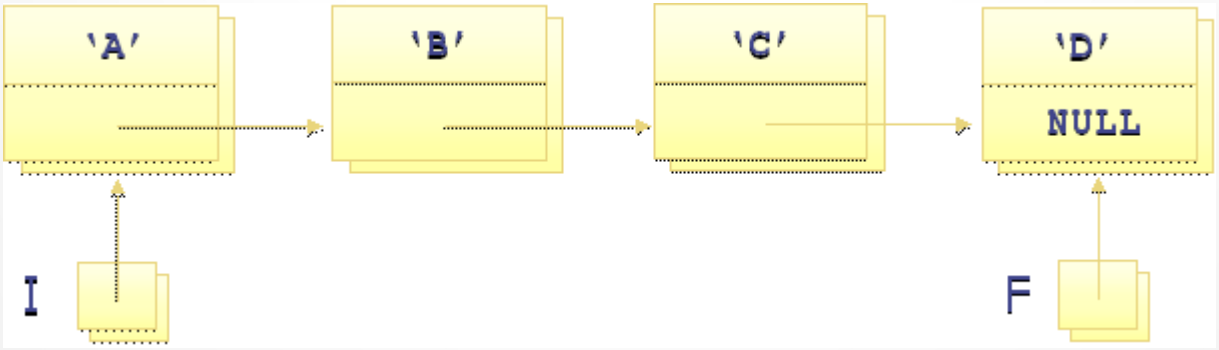
- Deve-se tomar cuidado apenas para que os ponteiros para início e final da final tenham valor NULL quando ela estiver vazia



Implementação Dinâmica

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

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

```
typedef struct {
   tNO *inicio;
   tNO *fim;
} Fila_Ligada;
```



Fila Ligadas

- Inserir Item (Última Posição)

```
int inserir (Fila_Ligada *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      pnovo->proximo = NULL;
      if (lista->fim != NULL){
           lista->fim->proximo = pnovo;
      } else {
           Lista ->inicio = pnovo;
      }
      lista->fim = pnovo;
      return 1;
    } else {
      return 0;
 }
```

Filas Ligadas

- Remover Item (Primeira Posição)

```
int remover (Fila_Ligada *lista){
   if (!vazia(lista)){
      tNO *paux = lista->inicio;
      if (lista->inicio == lista->fim) {
          lista->inicio = NULL;
          lista->fim = NULL;
      } else {
          Lista->inicio = lista ->inicio->proximo;
      }
      free(paux);
      return 1;
    }else {
        return 0;
    }
 }
```

Deques

Deques

- Deques são estruturas lineares que permitem inserir e remover de ambos os extremos

Deques

- inserir_inicio(D,x):
  - insere o elemento x no início do deque D. Retorna true se foi possível inserir false caso contrário
- inserir_fim(D,x):
  - insere o elemento x no final do deque. Retorna true se foi possível inserir false caso contrário
- remover_inicio(D):
  - remove o elemento no inicio de D. Retorna esse elemento. Retorna true se foi possível remover false caso contrário
- remover_fim(D):
  - remove o elemento no final de D. Retorna esse elemento. Retorna true se foi possível remover false caso contrário

Deques: Operações auxiliares

- primeiro(D):
  - retorna o elemento no início de D. Retorna true se o elemento existe false caso contrário
- ultimo(D):
  - retorna o elemento no final de D. Retorna true se o elemento existe false caso contrário
- contar(D):
  - retorna o número de elementos em D
- vazia(D):
  - indica se o deque D está vazio
- cheia(D):
  - indica se o deque D está cheio (útil para implementações estáticas)

Deques: Operações auxiliares

- Como deques requerem inserir e remover elementos em ambos os extremos
  - Implementação estática circular
  - **Implementação dinâmica duplamente encadeada**
- Nesses casos, as operações do TAD são O(1)

Implementação Estática Circular

- Aproveita-se a implementação de uma lista circular estática e acrescenta as duas operações que faltam:
  - (1) remover do fim;
  - (2) inserir no início

- As outras operações são as mesmas

Definição de Tipos

Fila circular

```
#define MAX 10

typedef struct {
      int valor;
} ITEM;

 typedef struct {
      ITEM itens[TAM];
      int fim;
      int inicio;
} Fila_Circular
```

Deque

```
#define MAX 10

typedef struct {
      int valor;
} ITEM;

 typedef struct {
      ITEM itens[TAM];
      int fim;
      int inicio;
} Deque
```

Implementação Estática Circular

```
int inserir_inicio(DEQUE_ESTATICA *deque, ITEM *item{
   if (!cheia(deque)) {
      deque->inicio = deque->inicio-1;
      deque->inicio = (deque->inicio+MAX-1)%MAX;
      if (deque->inicio < 0) {
         deque->inicio = MAX - 1;
      }
      //insiro o item
      deque->itens[deque->inicio] = *item;
     return 1;
    }
  return 0;
}
```

Implementação Estática Circular

```
int remover_fim(DEQUE_ESTATICA *deque, ITEM *item) {
    if (!vazia(deque)) {
      deque->fim = deque->fim-1;
      if (deque->fim < 0) {
        deque->fim = TAM - 1;
      }
     *item = deque->itens[deque->fim];
     //recupero o primeiro item
     return 1;
    }
  return 0;
}
```

Implementação Estática Circular

- Implemente uma fila dinâmica
- Implemente uma deque dinâmica
- Implemente um procedimento recursivo capaz de esvaziar uma fila
- Implemente um procedimento para inverter uma fila (o primeiro elemento se tornará o último e vice-versa)

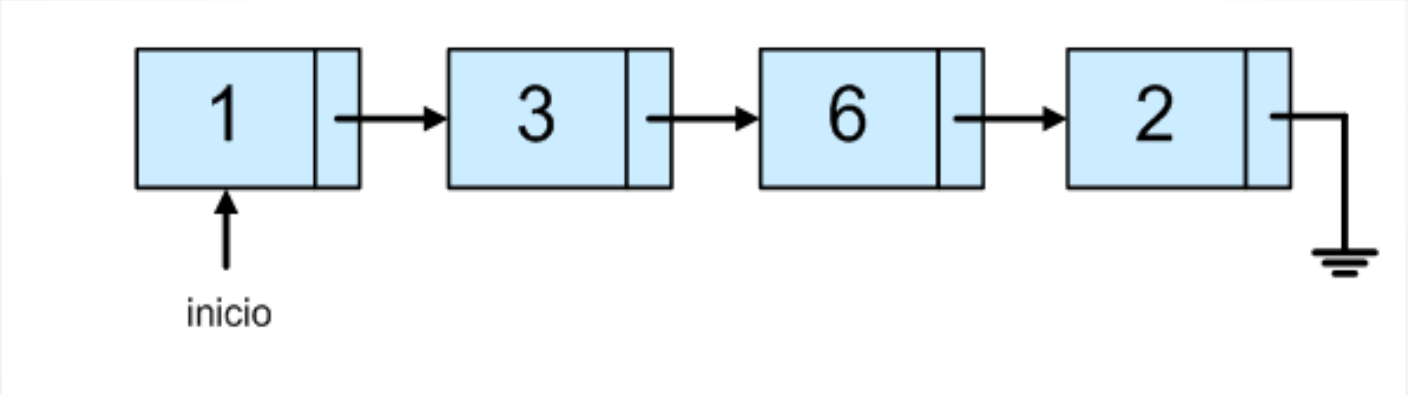
Listas Ligadas Circulares

Listas Ligadas Circulares

- O último nó indica o endereço do primeiro nó
- Não tem nem inicio e nem fim
- Uma pequena mudança na estrutura da lista ligada linear é realizada, isto é, o campo próximo no último nó contém um ponteiro de volta para o primeiro nó ao contrário de apontar para NULL
- Portanto a estrutura definida para uma lista ligada é a mesma coisa que a lista ligada linear dada a seguir
- Lista pode ser percorrida a partir de qualquer nó
- Lista com 1 só nó: o ponteiro do primeiro nó aponta para ele mesmo

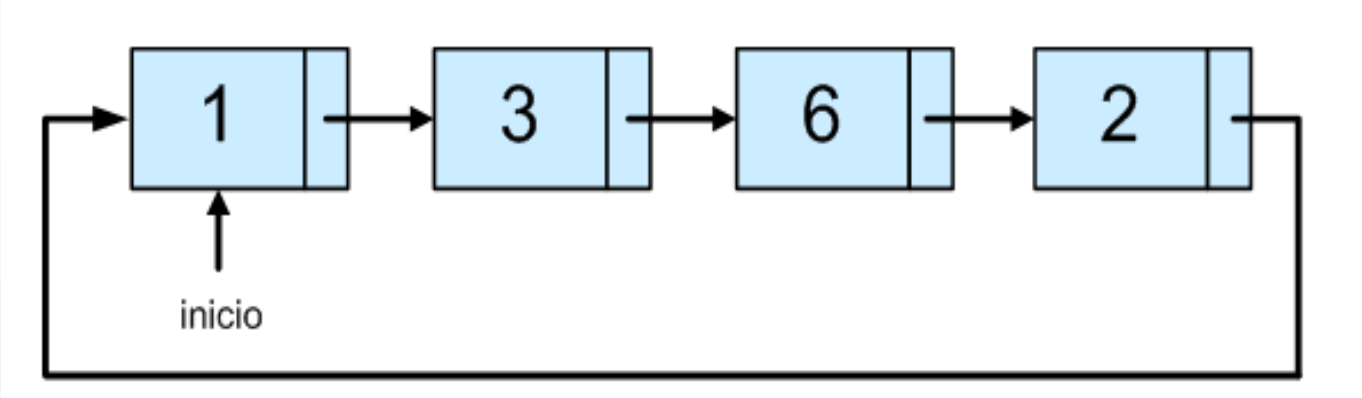
Introdução

- 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.

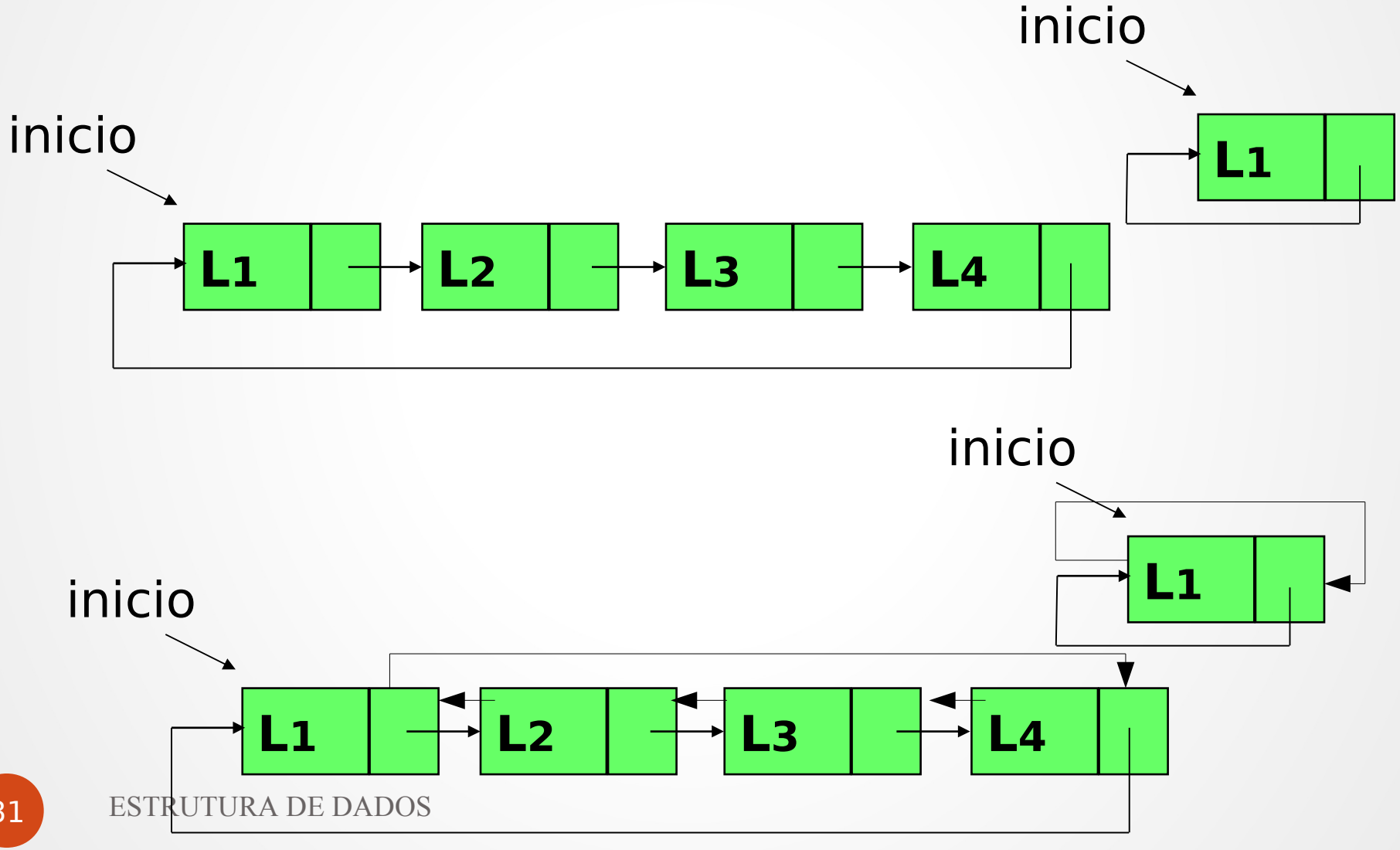


Introdução

- 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.



Listas Ligadas Circulares



Listas Ligadas Circulares

- **Criar uma lista**
- **Inserir novo nodo**
- **Remover um nodo**
- **Consulta um nodo**
- **Destruir lista**

Algoritmos

Semelhantes a Lista ligada simples

Introdução

- 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.
  - Todas as implementações apresentadas a seguir utilizam um ponteiro para o primeiro nó

Listas Ligada Circular

```
typedef struct {
    int chave;
    int valor;
} ITEM;
```

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

ideal

```
typedef struct {
    tNO *inicio;
} LISTA_Circular;
```

```
typedef struct {
    tNO *fim;
} LISTA_Circular;
```

Lista Ligada Circular

- As operações de criar e apagar a lista são simples

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

ou

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

Ponteiro para o início

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

Ponteiro para o fim

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

Lista Ligada Circular

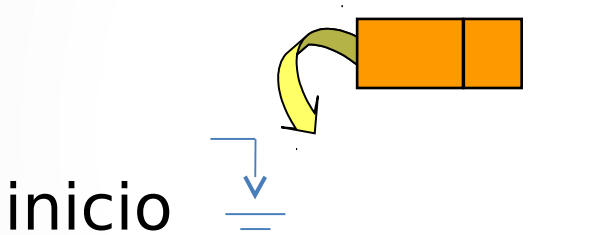
- Conta número elementos

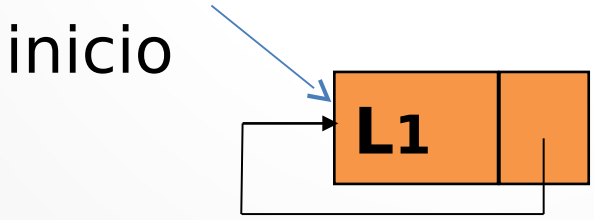
Ponteiro para o inicio

```
int conta (Lista_Ligada_Circular *lista){
   int cont;
   tNO *paux=lista->inicio
    if (lista->inicio==NULL){
         cont=0;
    }else{
        cont =1;
         while (paux->proximo!=lista->inicio){
            cont++;
            paux = paux->proximo;
         }
     }
    return cont;
 }
```

Inserção de novo nó

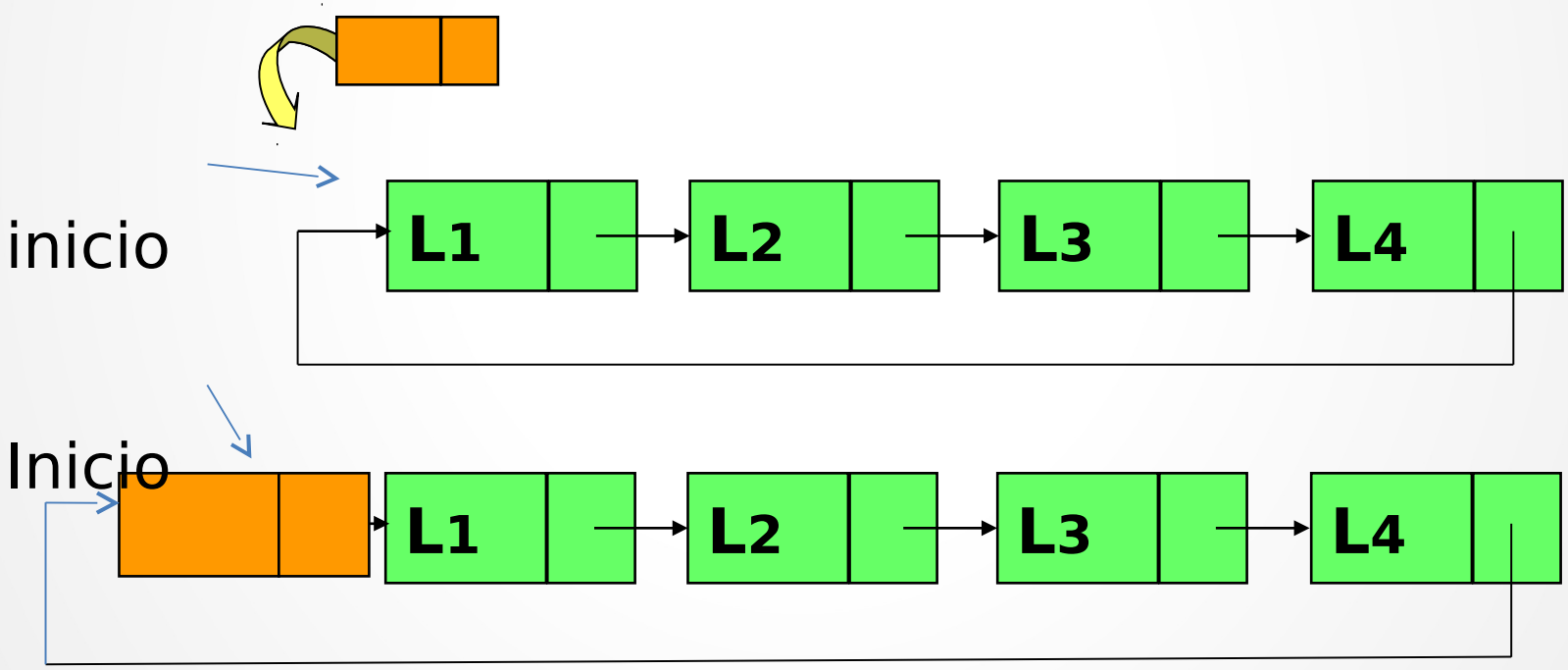
- Lista circular vazia





Inserção de novo nó em um dos extremos

- Inserção no inicio da lista



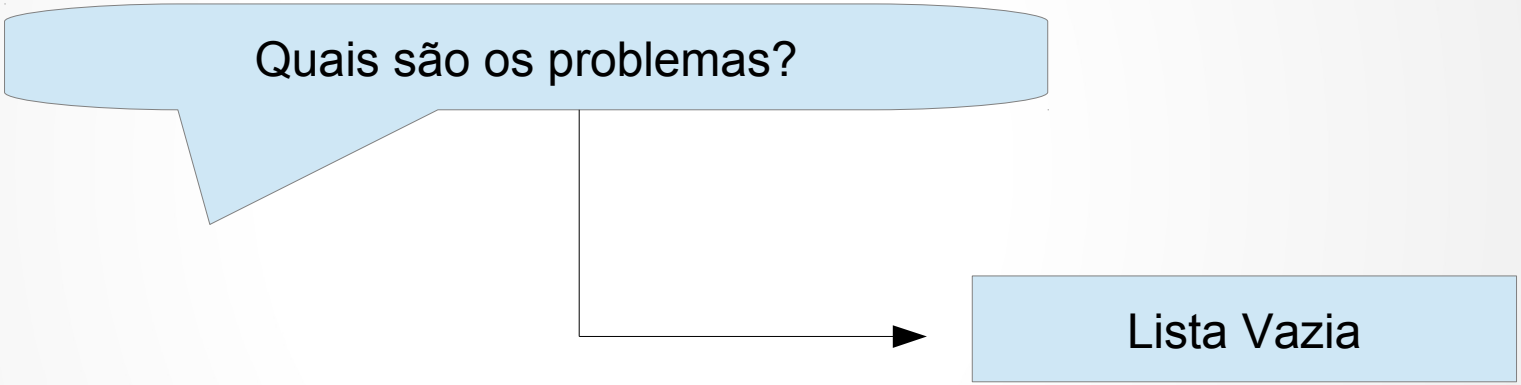
Lista Ligada Circular

- Inserir um Item no inicio

Quais são os problemas?

Lista Ligada Circular

- Inserir um Item no inicio



- Se ponteiro no inicio, encontrar o fim da lista

Lista Ligada Circular

- Inserir Item (Primeira Posição)

Ponteiro para o inicio

```
int inserir_inicio (Lista_Ligada_Circular *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo ==NULL) {
      pnovo->item = *item;
      if (lista->inicio==NULL){
          pnovo->proximo=pnovo;
      }else {
           //encontra a posição final
           tNO *paux=lista->inicio
           while (paux->proximo!=lista->inicio){
             paux = paux->proximo;}
           paux->proximo = pnovo;
           pnovo->proximo = lista->inicio;
      }
      lista->inicio = pnovo;
      return 1;
    } else {
      return 0;
    }
}
```

Lista Ligada Circular

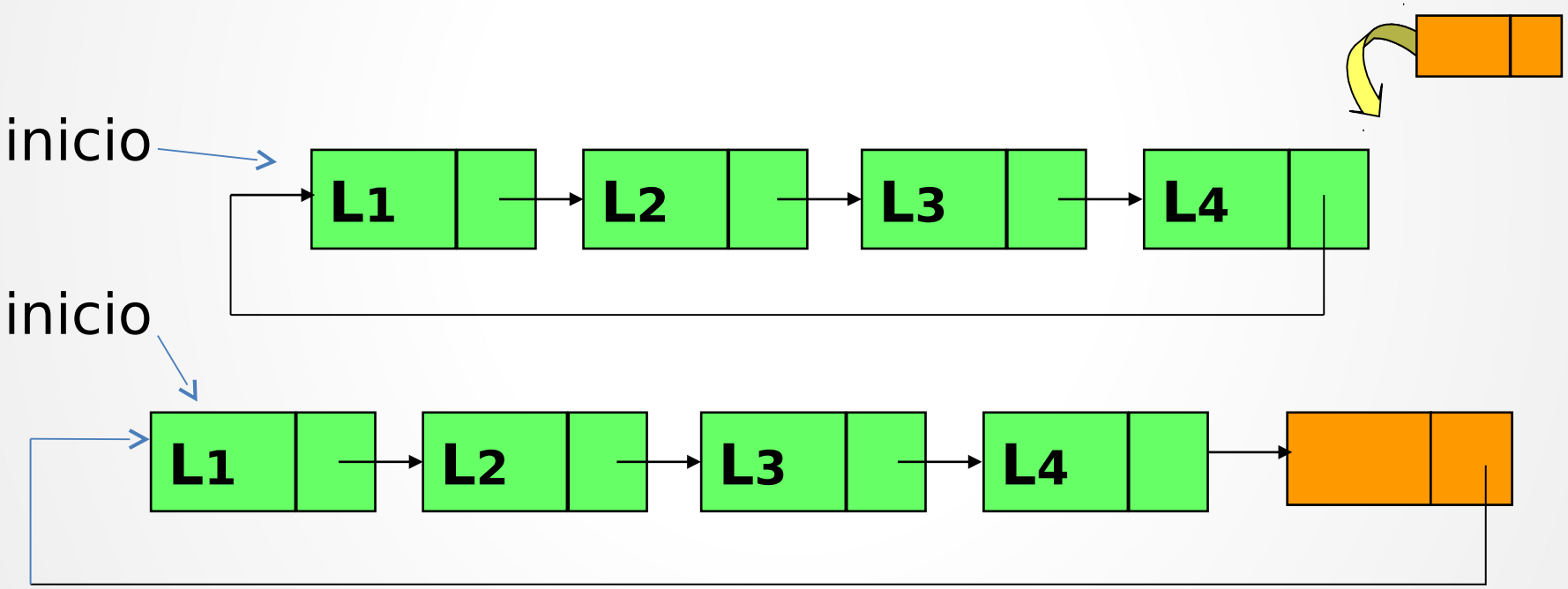
- Inserir Item (Primeira Posição)

Ponteiro para o fim

```
int inserir_inicio (Lista_Ligada_Circular *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      if (lista->fim==NULL){
          pnovo->proximo=pnovo;
          lista->fim=pnovo;
      }else {
           //encontra a posição final
           pnovo->proximo = lista->fim>proximo;
           lista->fim->proximo = pnovo;
      }
      return 1;
    } else {
      return 0;
    }
}
```

Inserção de novo nó em um dos extremos

- Inserção no fim da lista



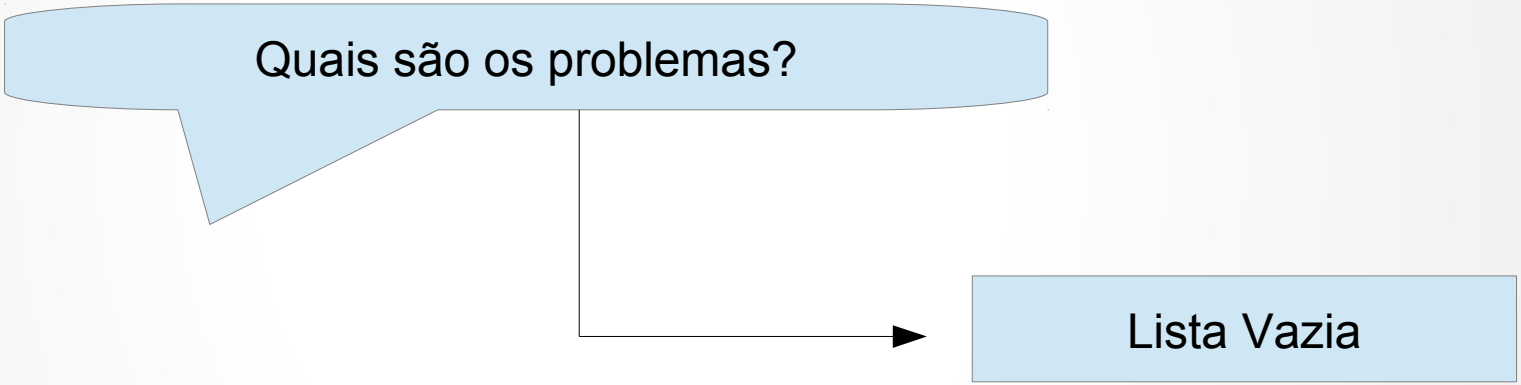
Lista Ligada Circular

- Inserir um fim no inicio

Quais são os problemas?

Lista Ligada Circular

- Inserir um Item no inicio



- Se ponteiro no inicio, copia-lo para outro ponteiro e ir para fim da lista

Lista Ligada Circular

- Inserir Item (Última Posição)

Ponteiro para o inicio

```
int inserir_fim (Lista_Ligada_Circular *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      if (lista->inicio==NULL){
          pnovo->proximo=pnovo;
          lista->inicio = pnovo;
      }else {
           //encontra a posição final
           tNO *paux=lista->inicio
          while (paux->proximo!=lista->inicio){
            paux = paux->proximo;}
           paux->proximo = pnovo;
           pnovo->proximo = lista->inicio;
      }
      return 1;
    } else {
      return 0;
    }
}
```

Lista Ligada Circular

- Inserir Item (Última Posição)

Ponteiro para o fim

```
int inserir_fim (Lista_Ligada_Circular *lista, ITEM *item){
   tNO *pnovo = (tNO *)malloc(sizeof(tNO));
   if (pnovo !=NULL) {
      pnovo->item = *item;
      if (lista->fim!=NULL){
          pnovo->proximo=pnovo;
      }else {
           //encontra a posição final
           pnovo->proximo = lista->fim->proximo;
           lista->fim->proximo = pnovo;;
      }
      lista->fim = pnovo;
      return 1;
    } else {
      return 0;
    }
}
```

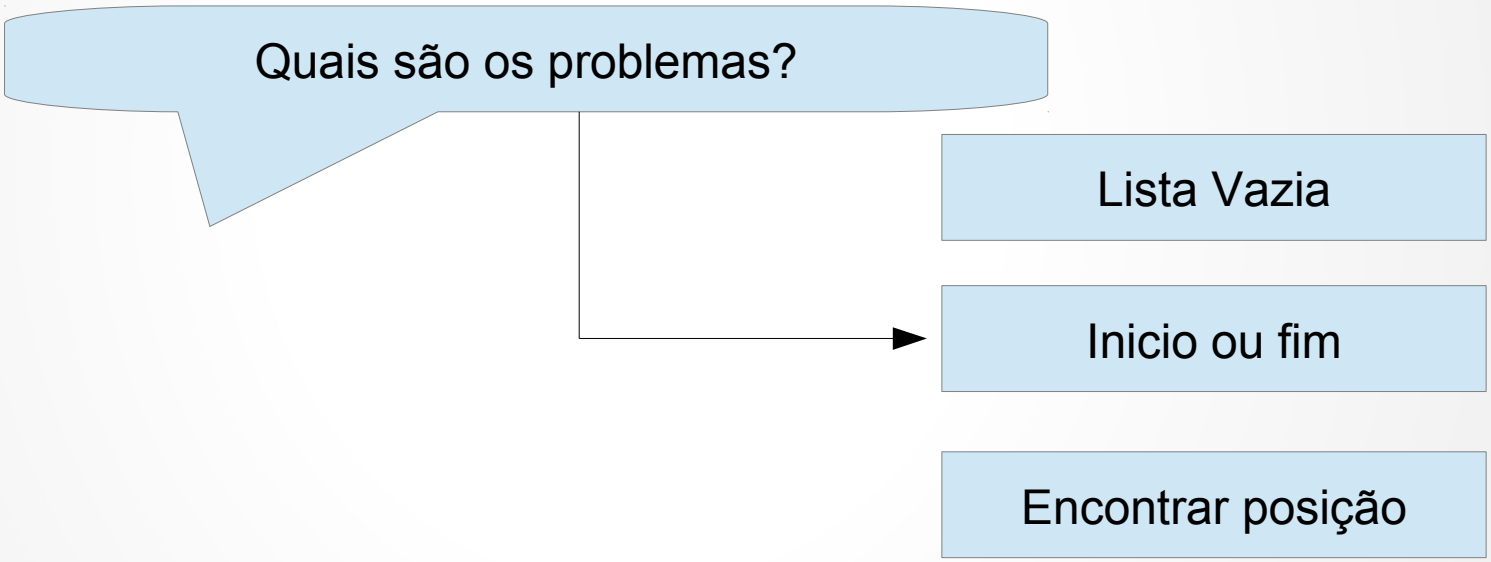
Lista Ligada Circular

- Inserir um item dada um posiçao

Quais são os problemas?

Lista Ligada Circular

- Inserir um Item no inicio



```
int inserir_posicao (Lista_Ligada_Circular *lista, int pos, ITEM *item){
    tNO *pnovo = (tNO *)malloc (sizeof(tNO)); //cria um novo nó
    tNO *patual, *pant;
    if (pnovo != NULL) {//verifica se existe memoria disponivel
       pnovo->item = *item;
       if (lista->inicio==NULL){ //lista vazia, difere de lista encadeada simples
           pnovo->proximo=pnovo;
           lista->inicio = pnovo;
       }else{
          if (pos==0) {//adiciona na primeira posicao
              pant = lista->inicio; //inicializa o ponteiro
              while (pant->prox!=lista->inicio)
                  pant=pant->prox;
             pnovo->proximo = lista->inicio;
             pant->proximo = pnovo;
             lista->inicio = pnovo; //atualiza o ponteiro inicio
           } else {
              patual = lista->inicio;
              //encontra a posição de inserção
              for (int i=0; i<pos; i++){
                   pant = patual;
                   patual = patual->proximo;}
              // faz as ligações para a inserção do novo elemento
             pnovo->proximo =patual;
             pant->proximo = pnovo;
            if ((pos%conta(lista))==1)
                lista->inicio=pnovo;
           }//fecha eles
       }
       return 1
     }
    else {
       return 0;}
},
```

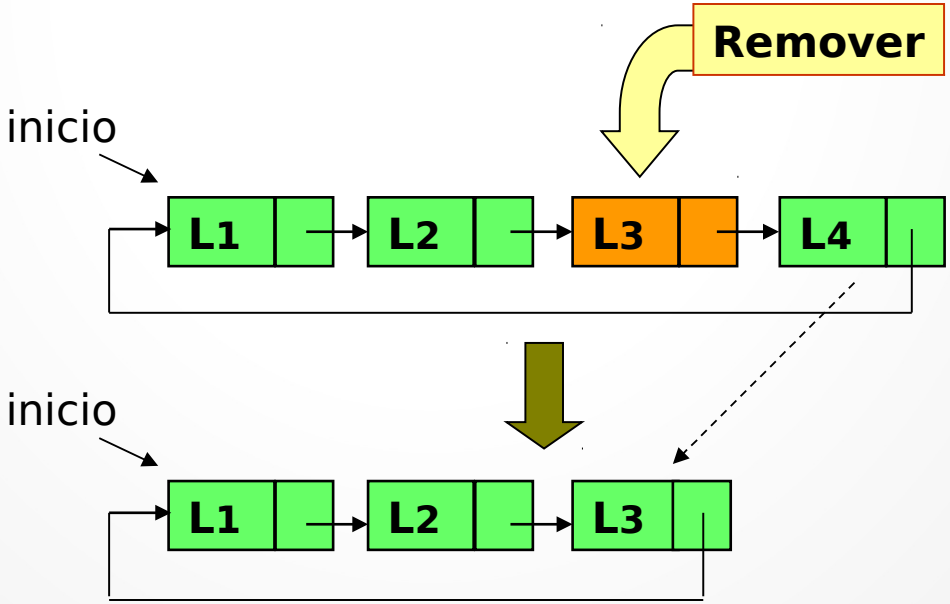
Ponteiro para o inicio

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

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

Ponteiro para o fim

Remoção nó

- Localizar posição do nodo
- Adequar encadeamentos
- Liberar nodo



Remoção nó

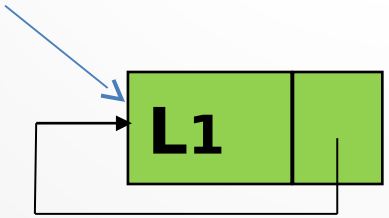
- Remover um nó de uma lista vazia

inicio

NÃO É POSSÍVEL

- Remover um nó de uma lista de único nó

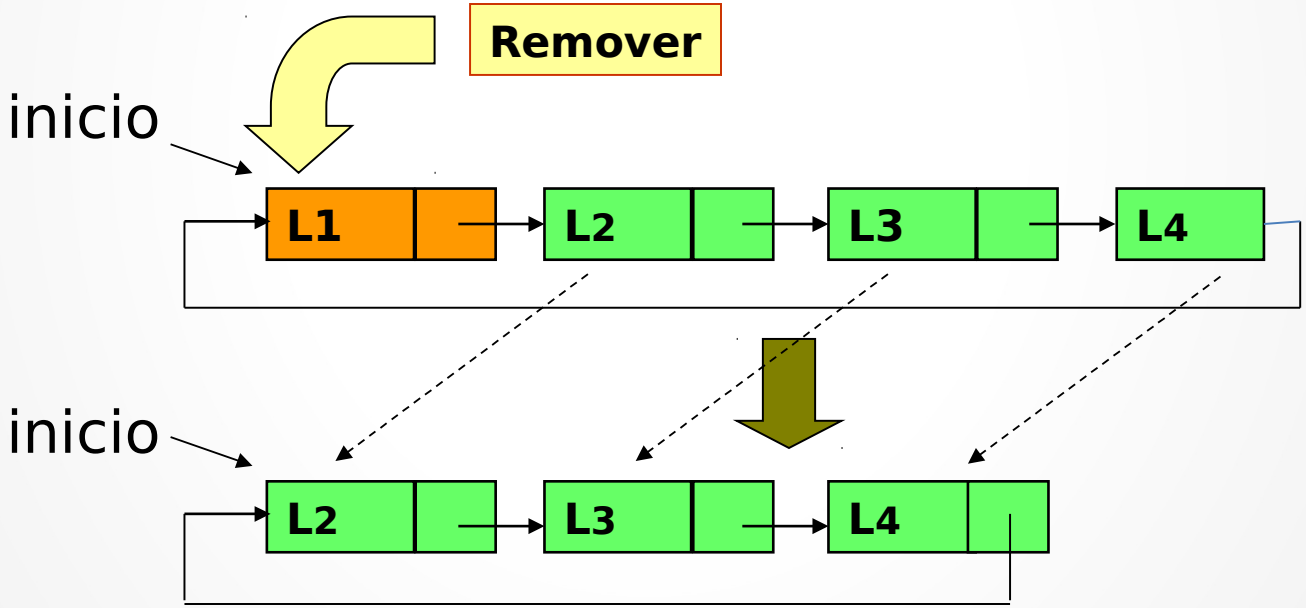
inicio



inicio

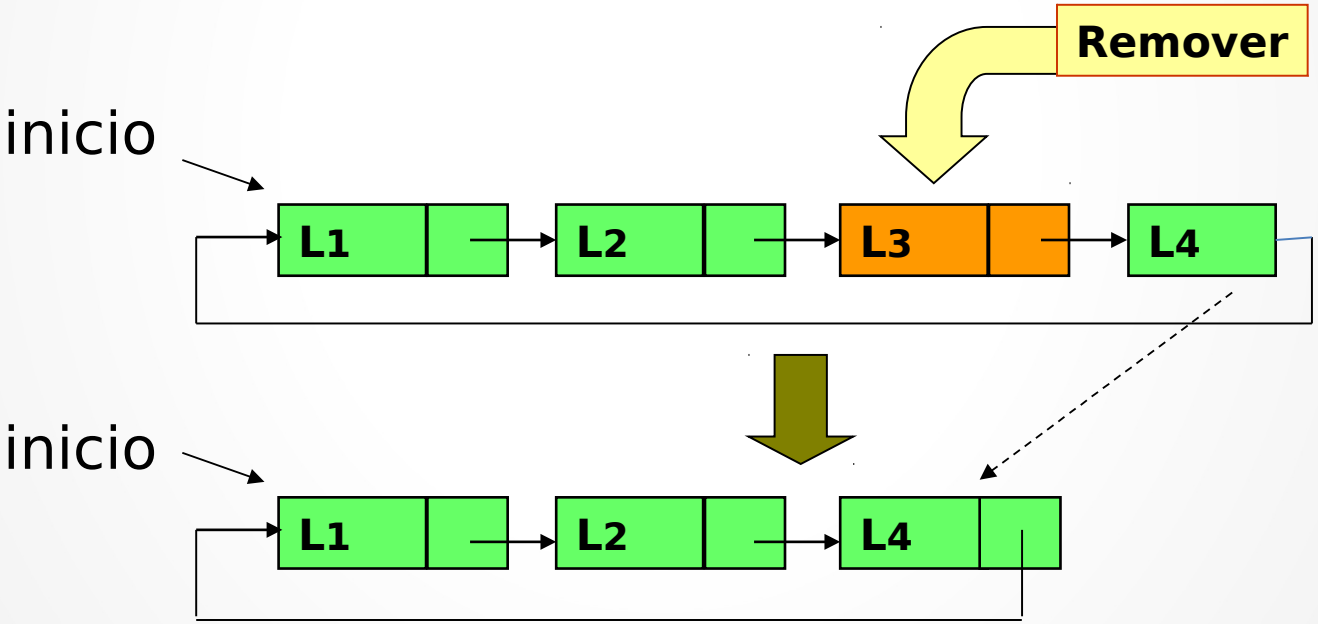
Remoção nó

- Remover o primeiro nó de uma lista



Remoção nó

- Remover um nó qualquer da lista



Remoção nó

- Caso 1: lista vazia
- Caso 2:nó a ser removido é o primeiro e único da lista
- Caso 3:nó a ser removida é o primeiro mas não é o único
- Caso 4:nó a ser removido é qualquer outro nó da lista

Ponteiro para o inicio

```
int remover_posicao (Lista_Ligada_Circular *lista, int pos){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
       int i;
       tNO *patual = lista->inicio;
       tNO *pant = lista->inicio;;
       //encontra a posição de remoção
       if (pos>0){
           for (i=0;i<pos;i++) {
               pant=patual
               patual = patual->proximo;
          }
       else if (pos==0){
            while (pant->prox!=lista->inicio)
                pant=pant->prox;
          }
       else{
           return 0;
       }
       if (patual==pant) {//remove o único elemento
              lista->inicio = NULL;
       }else{
            if (patual == lista->inicio ) //remove o primeiro item
                lista->inicio = patual->proximo;
            pant->proximo = patual->proximo;//remove um elemento do meio
       }
        free(patual); //remove o item da memoria
        return 1;
      }
    return 0;
  }
```

```
int remover_posicao (Lista_Ligada_Circular *lista, int pos){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
       int i;
       tNO *patual = lista->fim->proximo;
       tNO *pant = lista->fim;;
       //encontra a posição de remoção
        if (pos<0){
           return 0;
        }
        for (i=0;i<pos;i++) {
               pant=patual
               patual = patual->proximo;
        }
        if (paux==pant) {//remove o único elemento
              lista->fim = NULL;
       }else{
            if (patual == lista->fim ) //remove o ultimo item
                lista->fim = pant;
            pant->proximo = patual->proximo;//remove um elemento do meio
       }
        free(paux); //remove o item da memoria
        return 1;
      }
    return 0;
  }
```

Lista Ligada Circular

- Remover um item dada uma chave

Ponteiro para o início

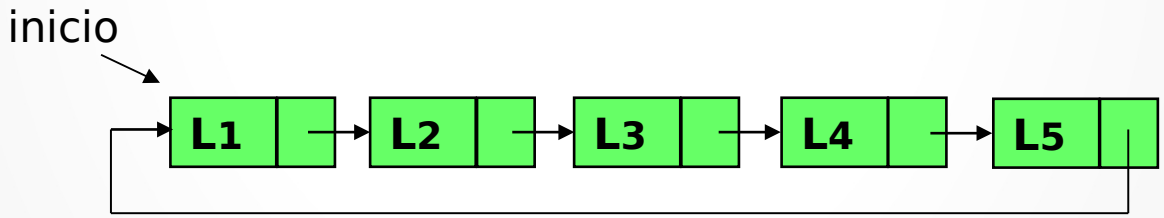
```
int remover_chave (Lista_Ligada_Circular *lista, int chave){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
       int i;
       tNO *patual = lista->inicio;
       tNO *pant ;
       if (patual->chave==chave) {//elemento é o primeiro da lista
           if (patual->proximo ==lista->inicio) {//apenas um elemento na lista
                 lista->inicio = NULL;
                 free(patual);
                 return 1;
           }else{//mais de um no na lista
                while (patual->proximo!=lista->inicio)
                      patual = patual->proximo;
                patual->proximo = lista->inicio->proximo;
                lista->inicio = patual->proximo;}
        }else { //não é o primeiro elemento
       //encontra a posição de remoção
          patual = patual->proximo;
          while (patual->item->chave!=chave&&patual!=lista->inicio){
              pant=patual;
              patual = patual->proximo;
          }
          if (patual== l->inicio)//não encontrou o elemento
              return 0;
          pant->proximo=patual->proximo;
          free(patual);
          return 1;
      }
    return 0;
  }
```

Ponteiro para o fim

```
int remover_chave (Lista_Ligada_Circular *lista, int chave){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
       int i;
       tNO *patual = lista->fim->proximo;
       tNO *pant = lista->fim;
       if (patual->chave==chave) {//elemento é o primeiro da lista
           if (patual->proximo ==lista->fim) {//apenas um elemento na lista
                lista->fim = NULL;
                 free(patual);
                 return 1;
           }else{//mais de um no na lista
                patual->proximo = lista->inicio->proximo;
                lista->inicio = patual->proximo;}
        }else { //não é o primeiro elemento
       //encontra a posição de remoção
          patual = patual->proximo;
          while (patual->item->chave!=chave&&patual!=lista->inicio){
              pant=patual;
              patual = patual->proximo;
          }
          if (patual== l->inicio)//não encontrou o elemento
              return 0;
          pant->proximo=patual->proximo;
          free(patual);
          return 1;
      }
     return 0;
  }
```

Consulta um nó na lista

- Iniciar sempre acessando o primeiro nó da lista
- Seguir acessando de acordo com o campo próximo
- Parar quando encontrar o primeiro elemento da lista



```
int consulta_kesimo (Lista_Ligada_Ciruclar lista, int pos, ITEM*
  item){
    if (!vazia(lista)) {//verifica se a lista não esta vazia
       if (pos<0) //posiçao invalida
          return 0;
       int cont = 0;
       tNO *patual = lista->inicio;
       while (patual->proximo!=lista->inicio&&cont<pos){
               patual->proximo = patual;
               cont++;}
       if (cont!=pos)//não encontrou posição
           return 0;
       else{
           *item = patual->item;
            return 1;}
    }
       return 0;
}
```

```
int apaga_lista (Lista_Ligada_Ciruclar lista){
   if (!vazia(lista)) {//verifica se a lista não esta vazia
      tNO *patual = lista->inicio;
      tNO *prem;
      while (patual->proximo!=lista->inicio){
         prem = patual;
         patual = patual->proximo;
         free(prem);
      }
      free(atual);
      lista->inicio = NULL;
  }
  return 0;
}
```