Universidade Federal do Ma
Curso de Ciência da Computação

**Estrutura de Dados**
**Prof. Anselmo C. de Paiva**
**Depto de Informática**

*Listas*



## *Limitações dos vetores*

- **Vetores**
  - **Simples,**
  - **Rápidos**

  ***Mas,***
  - **é necessário especificar o tamanho no momento da construção**
  - **Lei de Murphy: Construa um vetor com espaço para n elementos e você sempre chegará à situação em que necessita de n+1 elementos**

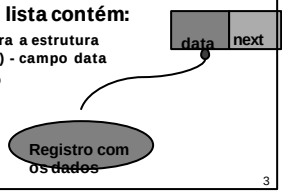




## *Listas Encadeadas*

- **Utilização flexível da memória**
  - **memória é alocada dinamicamente para cada elemento a medida que for necessária**
  - **cada elemento da lista inclui um ponteiro para o próximo**

**Lista Encadeada**
- **Cada item( tipo node) da lista contém:**
  - **o item de dado (ponteiro para a estrutura que realmente guarda os dados) - campo data**
  - **um ponteiro pra o próximo item da lista - campo next**

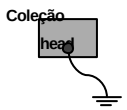







## *Listas Encadeadas*

- **A estrutura Coleção possui em substituição ao vetor de ponteiros para dados um ponteiro para a lista head**
  - **Inicializado com NULL**

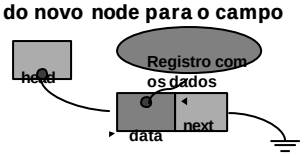







## *Listas Encadeadas*

- **Insere o primeiro elemento na lista**
  - **Aloca espaço para um *node***
  - **Atribui endereço do dado para o ponteiro data do node**
  - **Atribui NULL para o campo next do node**
  - **Atribui o endereço do novo node para o campo head da Coleção**



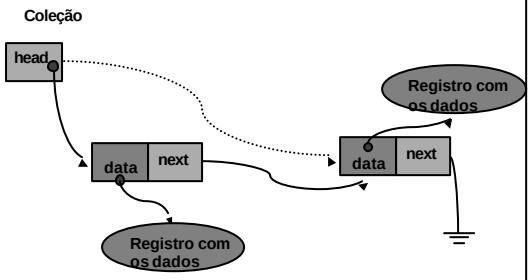





## *Listas Encadeadas*

- **Insere o segundo item**
  - **Aloca espaço para um node**
  - **Atribui endereço do dado para o ponteiro data do node**
  - **Atribui NULL para o campo next do node**
  - **Atribui o endereço do novo node para o campo head da Collection**
  - **Atribui o campo next do novo nodecom o valor de head**
  - **Atribui o endereço do novo node ao campo head da Collection**

## Listas Encadeadas








## Listas - Implementação operação ADD

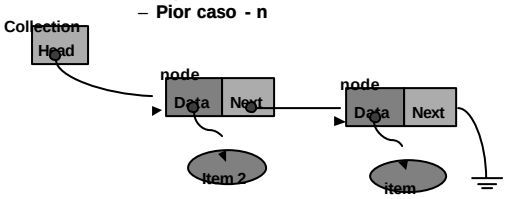
```
struct _node_ {
  void *item;
  struct _node_ *next;
} Node;
struct _collection_ {
    Node head;
    ……
}Collection;
int AddToCollection( Collection *c, void *item )
{
    Node *new = malloc( sizeof(Node) );
    new->item = item;
    new->next = c->head;
    c->head = new;
    return TRUE;
}
```





## Listas Ligadas

- **Tempo para inserção**
  - **Constante - independente do número de elementos na lista n**
- **Tempo de Consulta**
  - **Pior caso - n**



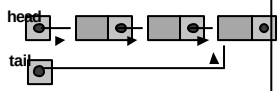





## Pilhas e Filas como Listas Ligadas

- **Pilha - Implementação mais simples**
  - **Adiciona e retira somente no início da lista**
  - **Last-In-First-Out (LIFO) semantics**
- **Fila**
  - **First-In-First-Out (FIFO)**
  - **Mantém um ponteiro para o final da lista**

```
struct _node_ {
    void *item;
    struct t_node *next;
}Node;
struct _collection_ {
    Node *head, *tail;
}Node;
```



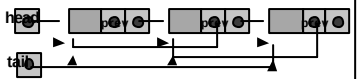





## Listas - Duplamente encadeadas

- **Podem ser percorridas nas duas direções**

```
struct _node_ {
  void *item;
  struct _node_ *prev, *next;
}Node;
struct _collection_ {
    Node *head;
}Collection;
```








## Operações sobre Listas Ligadas

1. **Criar lista vazia**
2. **Inserir primeiro elemento**
3. **Inserir no início de uma lista**
4. **Acessar primeiro elemento**
5. **Acessar último elemento**
6. **Tamanho da lista**
7. **Inserir valor v na posição p+1**

**1- Criação da lista vazia**

```
SLList *CreateCollection (void )
{
   SLList *l = (SLList*)
          malloc(sizeof(SLList));
   assert(l);
   l->numItens = 0;
   l->head = NULL;
   return l;
}
```

**2) Inserção do primeiro item**

```
void  AddFirstToCollection(
          SLList *l, void *item)
{
   Node *newNode= (Node *)

malloc(sizeof(Node));
   assert(newNode);
   newNode->data =item;
   newNode->next = NULL;
   l->head =newNode;
   l->numItens ++;
}
```





**3) Inserção no início**

```
void AddToCollection(
      SLList  *l, void *item) {
  Node *newNode  = (Node *)
         malloc(sizeof(Node ));
  assert( newNode);
  newNode->data = item;
  newNode ->next = l->head;
  l->head = newNode ->next;
  l->numItens++;
}
```

**4) Acesso ao primeiro elemento**

```
void *First( SLList *l) {
  return l->head->data;
}
```

**5) Acesso ao último elemento**

```
void *Last(SLList  *l) {
  Node *cur;
  if(l ->head == NULL ) {
    return NULL;
  }
  cur = l->head;
  while(cur ->next != NULL ) {
        cur  =  cur ->next;
  }
  return cur ->data;
}
```





**6) Qual o número de elementos ?**

```
int NumElem( SLList *l )
{
   if( l->head  == NULL ) {
     return 0;
   } else {
     current = l;
    nElem := 1;
    while(current->next != NULL)
    {
       nElem++;
       current = current->next;
    }
 }
```

**7) Inserção do valor v depois do elemento apontado pork**

```
void InsertAfter( Node * k,
                  void *item)
{
   Node *newNode;
   newNode = (Node)
    malloc(sizeof(Node));
   newNode ->data = item;
   newNode->next = k->next;
   k->next = newNode;
}
```





- Eliminar elemento da posição k+1
- Eliminar primeiro elemento
- Eliminar valor v
- Inserir valor v antes do elemento apontado por p
- Criar uma lista com registros numerados
- Eliminar sucessor de p
- Imprimir recursivamente



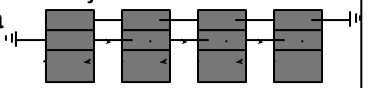


- **Uma lista L de elementos do tipo T é uma sequência de elementos T, em conjunto com as seguintes operações:**
  - **Construir uma lista vazia**
  - **Determinar quando a lista está vazia ou não**
  - **Determinar quando a lista está cheia ou não**
  - **Encontrar o número de elementos na lista**
  - **Limpar uma lista para torná-la uma lista vazia**
  - **Inserir um elemento em uma posição especificada**
  - **Remover um elemento de uma posição especificada**
  - **Substituir um elemento em uma posição especificada**
  - **Percorrer a lista, realizando uma certa operação em cada entrada da lista**





- **Lista linear encadeada onde cada elemento sabe onde está localizado os seus dois vizinhos. Possui um ponteiro para o próximo(*next*) e para o anterior(*previous)***
- **Permite movimentação bidirecional ao longo da lista**



## Operações sobre LDE

1. Criar lista vazia
2. Inserir numa das extremidades
3. Inserir após um elemento da lista
4. Inserir antes de um elemento da lista
5. Remover um elemento da lista
6. Localizar um elemento da lista





## Implementação das Operações

```
typedef struct _dlnode_
{
 struct _dlnode_ *next,
                 prev;
 void * data;
}DLNode;

typedef struct _dllist_
{
     DLNode *head;
}DLList
```

```
DLList *Create (void ) { /* 1- Cria lista vazia */
   DLList *l = (DLList *)malloc(sizeof(DLList));
   assert(l);
   l->head = NULL;
   return l;
}
/* 2) Inserção no início da lista */
void AddInListBegin(DLList *l, void *item) {
  DLNode *newNode=(DLNode *)
                  malloc(sizeof (DLNode) );
   assert(newNode);
   newNode->prev= NULL;
   newNode->data =item;
   if(l->head != NULL) { l->head->prev = newNode; }
   newNode->next = l->head;
   l->head = newNode;
}
```





## Implementação das Operações

```
void AddInListEnd(DLList *l, void *item) { // Inserção no final
   DLNode *newNode = (DLNode *)malloc(sizeof(DLNode));
   DLNode *last;
   assert(newNode);
   last=GetLastNode(l);
   newNode->data = item;
   newNode->next = NULL;
   newNode->prev = last;
   if (last != NULL) { last->next = newNode;
   } else {
     l->head = newNode;
   }
}
```





## Implementação das Operações

```
5) Remover um elemento da lista
int Remove(DLList *l, DLNode *elm)
{
    DLNode *prev=elm->prev,
           *next=elm->next;
    if (prev != NULL) {
       prev->next = next;
    } else {
      l->head = next;
    }
    if(next!=NULL) {
      next->prev= prev;
    }
    free(elm);
}
```





## Implementação das Operações

```
6) Inserir após um elemento da lista
int InsAfterElm( DLList *l, DLNode *ref, void *data ) {
   DLNode *newNode = (DLNode *)malloc(sizeof(DLNode ));
   DLNode *next=ref->next,  *prev=ref;
   newNode->data = data;
   newNode->next = next;
   newNode->prev = prev;
   if(prev != NULL) {
      prev->next = newNode;
   } else {
     l->head = newNode;
   }
   if( next!=NULL) { next->prev = newNode; }
 }
```



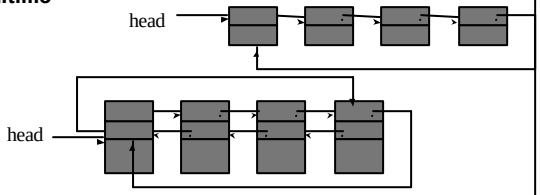


## Implementação das Operações

```
7) Inserir antes de um elemento da lista
int InsBefElm( DLList *l, DLNode *ref, void *data ) {
   DLNode *newNode = (DLNode *)malloc(sizeof(DLNode ));
   DLNode *next=ref, *prev=ref->prev;
   newNode->data = data;
   newNode->next = next;
   newNode->prev = prev;
   if(prev != NULL) {
      prev->next = newNode;
   } else {
     l->head = newNode;
   }
   if( next!=NULL) {  next->prev = newNode; }
 }
```

## Listas Encadeadas Circulares

- **O último elemento recebe o endereço do primeiro (Lista simplesmente encadeada)**
- **O campo next do último elemento aponta para o primeiro e o campo previous do primeiro aponta para o último**








## Listas Duplamente Encadeadas

- **O que muda??**
  - **A forma de identificar o último elemento da lista - agora é o elemento cujo campo next é igual a head.**
  - **Não precisa entrar na lista somente pelo head, agora basta saber o endereço de um Node e pode percorrer a lista. Nesse caso para qdo o campo next for igual ao Node fornecido**
- **Exercícios**
  - **Escreva os algoritmos para as seguintes operações em listas circulares duplamente encadeadas**
    - **Busca de elemento**
    - **Remoção de um elemento**
    - **Inserção após um elemento.**