ESTRUTURA DE DADOS

Prof. Clodoaldo A. M. Lima

12 de Março de 2013

Listas Lineares

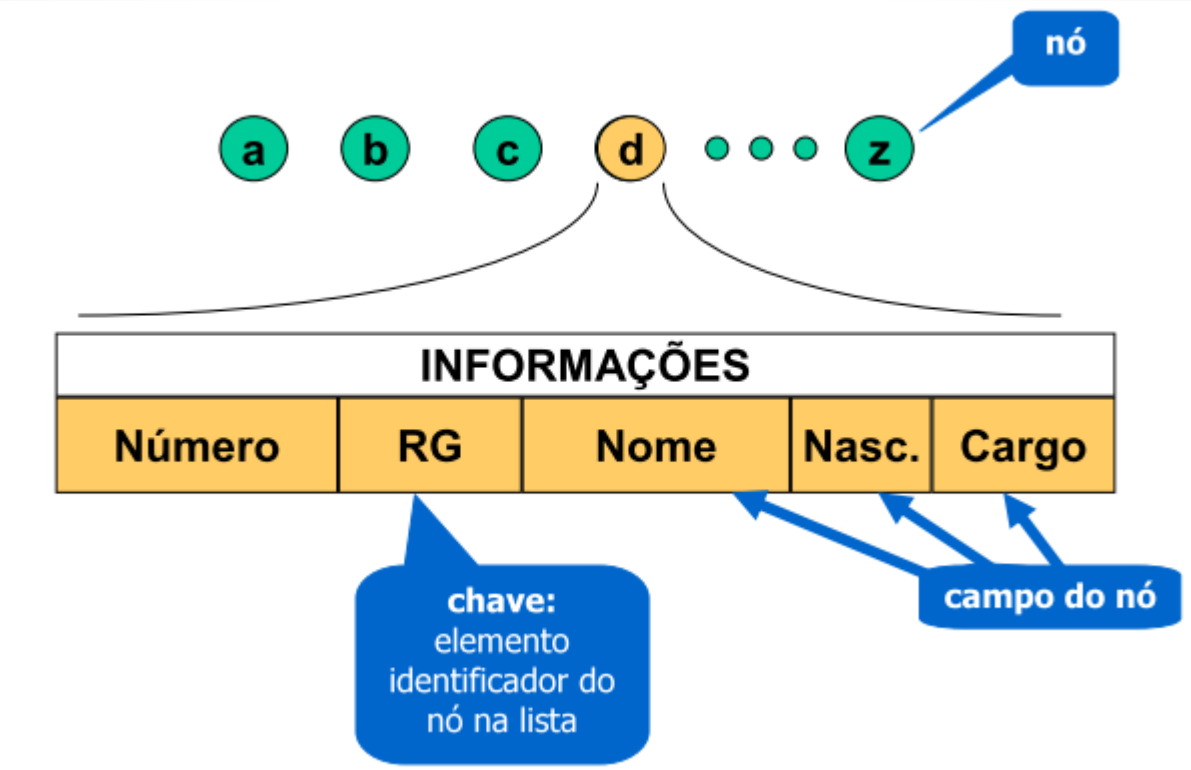
Listas Lineares

- A Lista Linear é a estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem existente entre eles.
- Uma lista linear, é um conjunto de n>= 0 nós L[1], L[2], ..., L[n] , onde:
  - Se n>0, L[1] é o primeiro nó,
  - Para 1 < k <= n, o nó L[k] é precedido por L[k-1].
- Observe que existe uma ordenação implícita: se n > 0 temos que:
  - L[1] é o primeiro nó da lista.
  - L[n] é o último nó da lista.
  - se 1 < k < n L[k] é precedido por a(k-1) e seguido por a(k+1).
  - se n = 0 dizemos que a lista é vazia.

Sequência de nós

Listas Lineares

- Estrutura interna é abstraída



Listas Lineares: Operações

- Criar uma lista vazia
- Verificar se uma lista está vazia
- Obter o tamanho da uma lista
- Obter/modificar o valor do elemento de uma determinada posição na lista
- Obter a posição de elemento cujo valor é dado

Listas Lineares: Operações

- Inserir um novo elemento após (ou antes) de uma determinada posição na lista
- Remover um elemento de uma determinada posição na lista
- Exibir os elementos de uma lista
- Concatenar duas listas

Listas Lineares: Classificação

Listas Lineares

Listas Lineares Gerais

- SEM restrição para inserção e remoção de elementos

Listas Lineares Particulares (Pilhas, Filas)

- COM restrição para inserção e remoção de elementos

Listas Lineares: Classificação

Listas Lineares

Listas Lineares Gerais

- SEM restrição para inserção e remoção de elementos

Listas Lineares Particulares (Pilhas, Filas)

- COM restrição para inserção e remoção de elementos

Listas Lineares: Classificação

- Listas lineares gerais:

  – A inclusão e remoção de elementos pode ser realizada em qualquer posição da lista. Essas listas não apresentam nenhuma restrição de acesso

- Casos particulares de listas:

  – **Pilha:** Inserções e remoções são realizadas em apenas um extremo (topo);

  – **Fila:** inserções realizadas em um extremo (fim) e remoções em outro (início).

Listas Lineares: Implementações

- Várias estruturas de dados podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares.

- As duas representações mais utilizadas são as implementações por meio de **arranjos** e de **apontadores**.

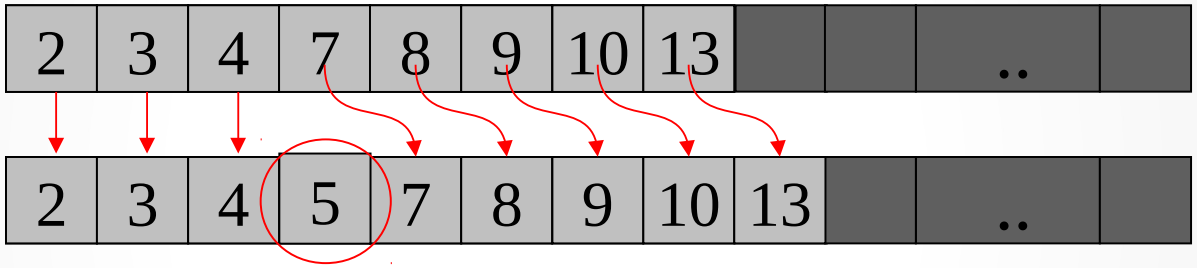
Implementação de Listas: Arranjos

- Os itens da lista são armazenados em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.
- Os itens são armazenados em um *array* de tamanho suficiente para armazenar a lista.
- O i-ésimo item da lista está armazenado na i-ésima posição do array, $1 \leq i \leq n$.


```
#define MAX 10

typedef int telem;
typedef struct {
        telem v[MAX];
        int n;
 } tlista;
```

Inserção em Listas: Arranjos

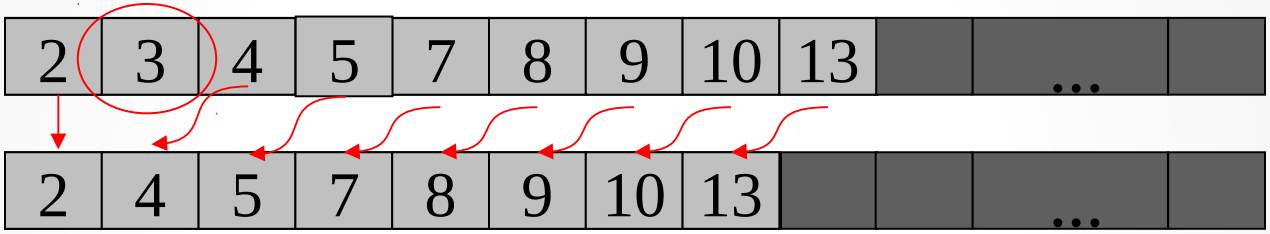
- A inserção de um novo item no meio da lista requer um deslocamento para a direita de todos os itens localizados após o ponto de inserção.




```
int inserir (tlista *L, int pos, telem dado) {
      int i;

       /*  lista cheia ou posição inválida */
      if ( (L->n == MAX - 1) || (pos > L->n + 1) )
          return (0);

      for (i=L->n; i>=pos; i--)
        L->v[i + 1] = L->v[i];

      L->v[pos] = dado;
      (L->n)++;
      return (1);
   }
```

Remoção em Listas: Arranjos

- A remoção de um item da lista requer um deslocamento para a esquerda de todos os itens localizados após o ponto de remoção.



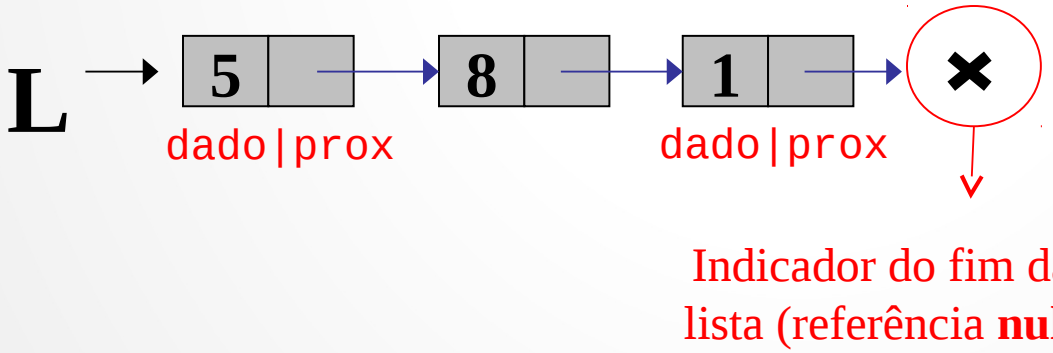

```
int remover (tlista *L, int pos, telem *dado) {

  int i;

/* posição inválida */
  if ( (pos > L->n) || (pos < 0) ) return (0);

  *dado = L->v[pos-1];

  for (i=pos; i< (L->n); i++)
     L->v[i] = L->v[i + 1];

  (L->n)--;
  return (1);
}
```

Listas usando Arranjos: Vantagens e Desvantagens

- **Vantagem:**
  - Acesso direto indexado a qualquer elemento da lista.
  - Economia de memória (não utiliza apontadores).
- **Desvantagens:**
  - Custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso;
  - Tamanho máximo da lista pré-estimado (definido em tempo de compilação).

Implementação de Listas: Apontadores

- A lista é constituída de nós.
- Cada nó contém um item da lista e um apontador para o nó seguinte.
- Toda lista possui um apontador externo **L** que aponta para o primeiro nó da lista (o início da lista)



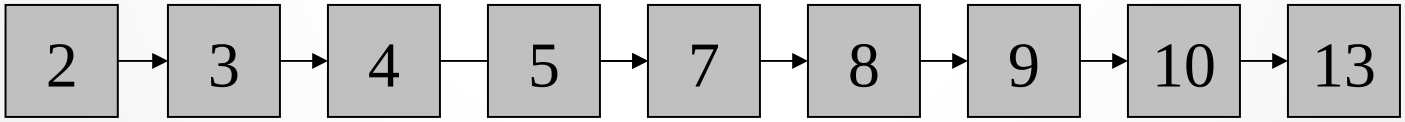

```
typedef  int telem;

typedef struct no {
            telem dado;
            struct no* prox;
} tno
typedef  tno*  tlista
```

Implementação de Listas: Apontadores

- Permite utilizar posições não contíguas de memória.
- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.

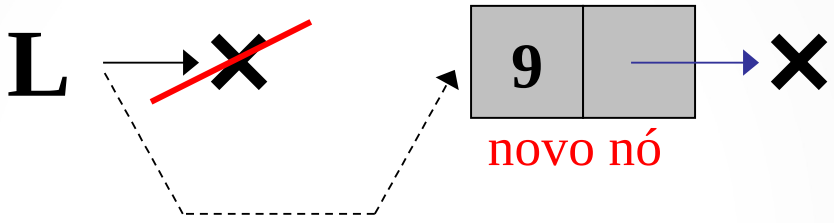
- O acesso é sequencial, ou seja, a busca inicia por um nó, depois vai pra outro nó, e assim por diante, até que se encontre o nó procurado.

- Para inserir um elemento, basta criar um nó, encontrar a posição desejada e inseri-lo;



Inserção em Listas: Apontadores

- Se a lista estiver vazia:

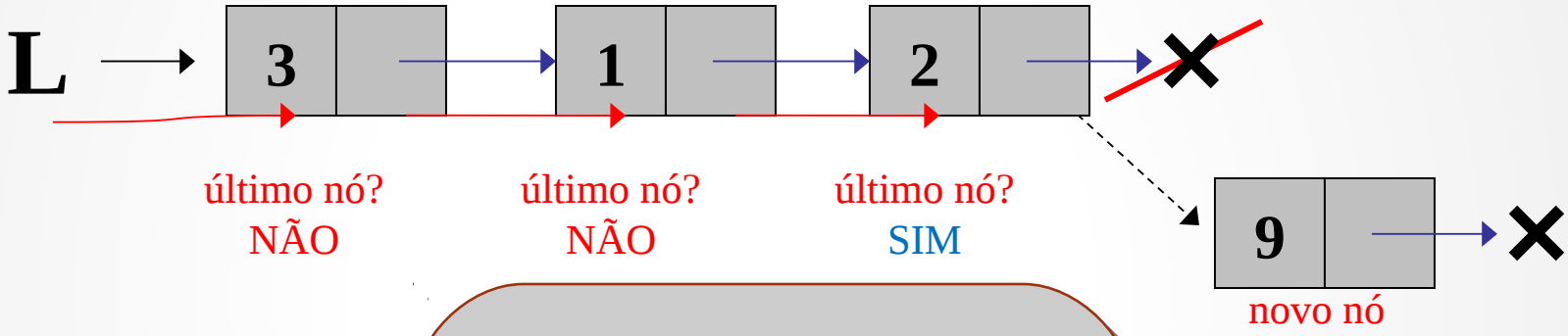


- Inserindo um novo elemento, teremos:

```
void inserir(tlista *L, telem valor){
   tlista  novo;
   novo = (tlista) malloc(sizeof(tno));
   novo->dado = valor;
   novo->prox = NULL;
   *L = novo;
}
```

Inserção em Listas: Apontadores

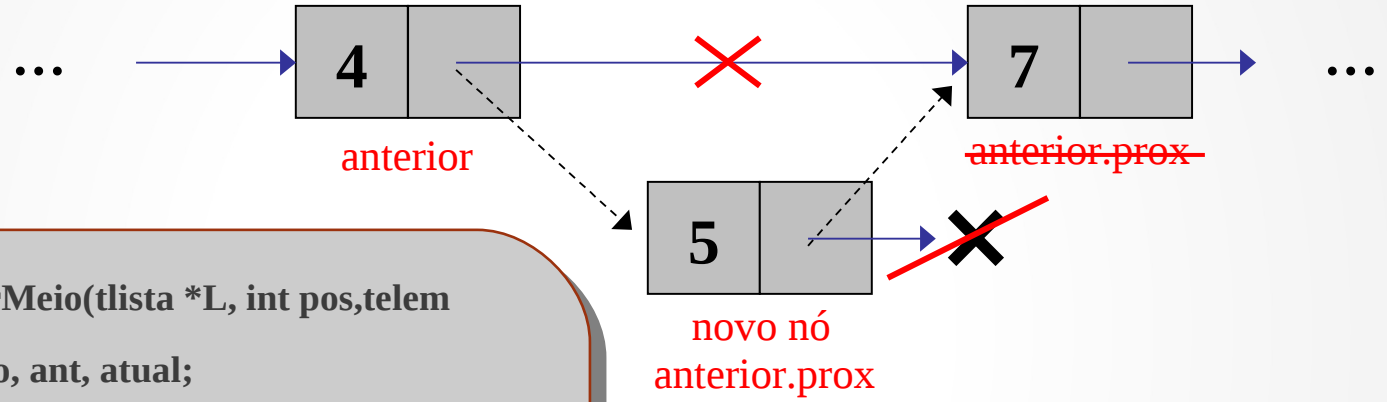
- Se a lista não estiver vazia, para inserir no fim da lista teremos:




```
void inserirFim(tlista *L, telem valor){
   tlista novo, ant, atual;

  ant = NULL;
  atual = *L;

while (atual  != NULL){
      ant = atual;
      atual = atual->prox;
   }

  novo = (tlista) malloc(sizeof(tno));
  novo->dado = valor;
  novo->prox = NULL;

   ant->prox = novo;
```

Inserção em Listas: Apontadores

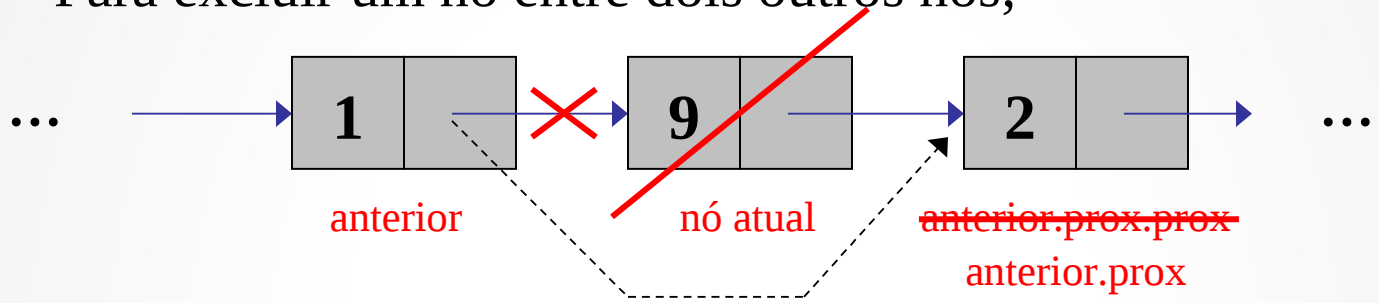
- Para inserir um nó entre dois outros nós:




```
void inserirMeio(tlista *L, int pos,telem
valor){
   tlista novo, ant, atual;
   int n;
   n = 1;
   atual = *L;

   while ((n<=pos-1) && (atual!=NULL)) {
      ant =  atual;
      atual =  atual->prox;
      n++;
   }
  novo = (tlista) malloc(sizeof(tno));
  novo->dado = valor;
  novo->prox = ant->prox;
  ant->prox =  novo;
}
```

Remoção em Listas: Apontadores

- Para excluir um nó entre dois outros nós;




```
int remover(tlista *L, int pos){
    tlista ant, atual;
    int n;

   if (vazia(*L)) return 0; /* erro: lista vazia */
   atual = *L;
   n = 1;
   while ((n<=pos-1) && (atual!=NULL)) {
      ant =  atual;
      atual = atual->prox;
      n++;
   }
    ant->prox = atual->prox;
   free(atual);
return 1;
}
```

Listas usando Apontadores: Vantagens e Desvantagens

- **Vantagens:**
  - Permite inserir ou retirar itens do meio da lista a um custo constante (importante quando a lista tem de ser mantida em ordem).
  - Bom para aplicações em que não existe previsão sobre o crescimento da lista (o tamanho máximo da lista não precisa ser definido a *priori).*
- **Desvantagem:**
  - Utilização de memória extra para armazenar os apontadores.