SCC-223 Estruturas de Dados I

Lista Linear Encadeada
(Dinâmica)

Profa. Elaine Parros Machado de Sousa

ICMC USP SÃO CARLOS

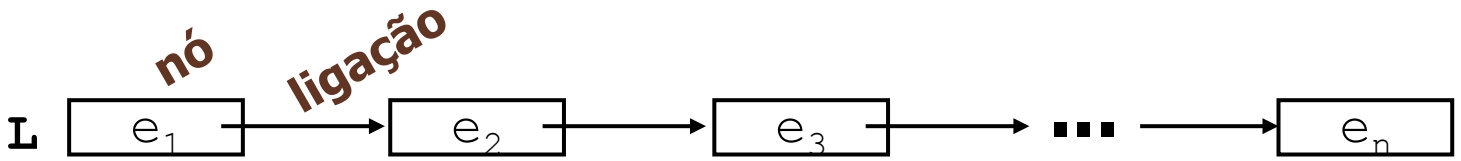
Aula baseada em material dos professores Adenilso Simão e Rudinei Goularte.

Grupo de Bases de Dados e Imagens
USP - São Carlos

Relembrando...

- **Lista Encadeada** ⇒ definida como uma sequência (lógica) de **nós encadeados**

ORGANIZAÇÃO LÓGICA

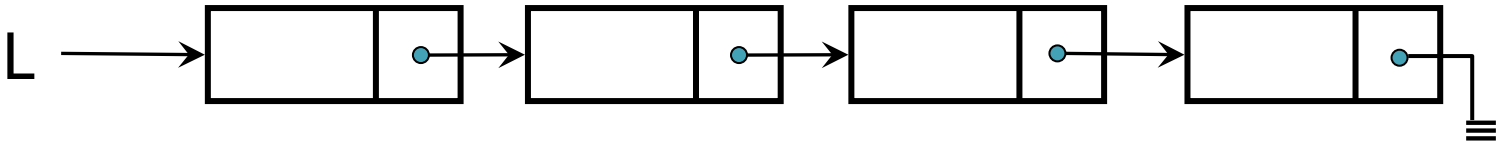


- Lista Encadeada Estática
  - *arrays*
- Lista Encadeada Dinâmica
  - ponteiros

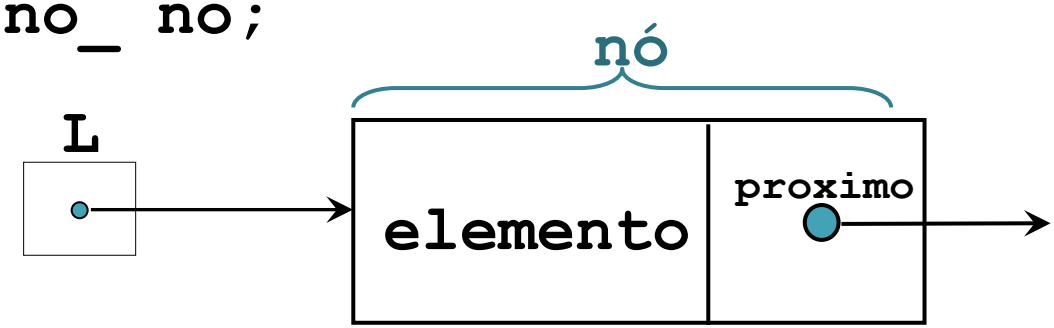
Lista Encadeada Dinâmica

- Não é necessário prever o tamanho da lista
  - o espaço disponíveis para **inserção** corresponde a toda a memória disponível para o programa durante a execução
  - **alocação dinâmica de nós**
- Campo de **ligação** dos nós ⇒ **endereços** reais da memória principal
  - linguagem C ⇒ **ponteiros**

Lista Encadeada Dinâmica



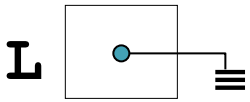

```
/* estrutura básica do nó */
struct no_
{
    tipo_elem elemento;
    struct no_ *proximo;
};
typedef struct no_ no;
no *L;
```



Lista Encadeada Dinâmica

- Lista vazia:

```
L == NULL;
```

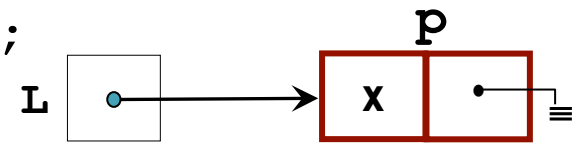


- Último elemento:

```
...-> proximo == NULL;
```

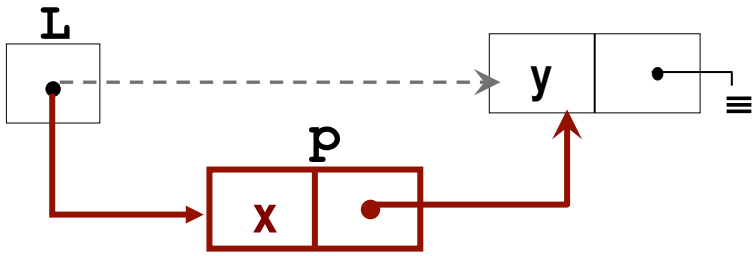
- Inserção do 1º elemento:

```
p = (no*) malloc(sizeof(no));
p->elemento = x;
p->proximo = NULL;
L = p;
```



- Inserção no início da lista.

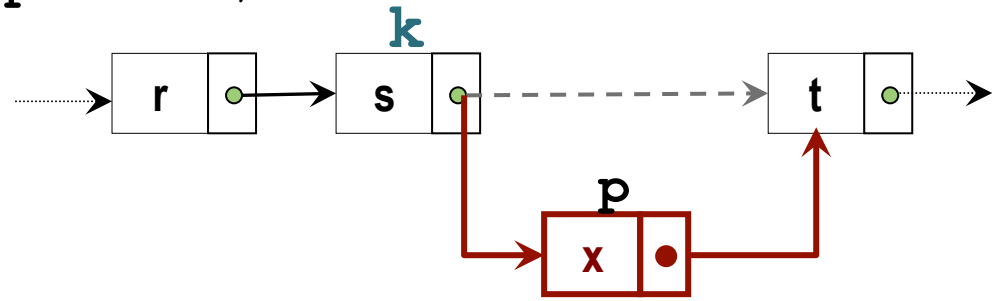
```
p =(no*) malloc(sizeof(no));
p->elemento = x;
p->proximo = L;
L = p;
```



Lista Encadeada Dinâmica

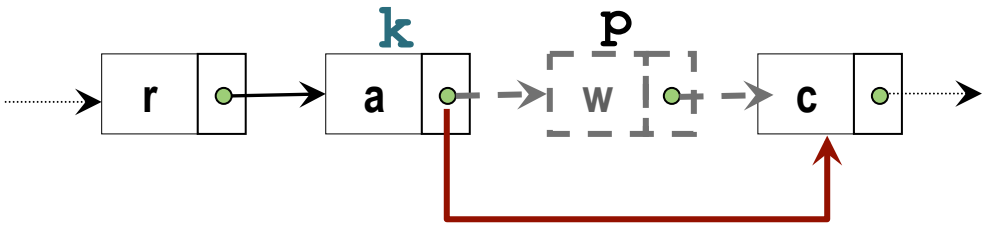
- Inserção após o k-ésimo elemento

```
p = (no*)malloc(sizeof(no));
p ->elemento = x;
p->proximo = k->proximo;
k->proximo = p;
```



- Remoção após o k-ésimo elemento

```
p = k->proximo;
k->proximo = p->proximo;
free(p);
```



Lista Encadeada Dinâmica

- Percorrer a lista – visitar todos os nós

```
p = L;
while (p != NULL)
  p = p->proximo;
```

- ....

Exemplo de Implementação

- **TAD Lista Não Ordenada (aulas anteriores) – implementação Encadeada Dinâmica**
  - **mesma definição**
    - tipo de dado **`LISTA`**
    - funções com os mesmos nomes e parâmetros
  - **implementação diferente**
    - organização como lista encadeada dinâmica

TAD Lista Não Ordenada – Exemplo de Definição da Interface

Arquivo Lista.h

- ✓ o mesmo criado para Lista Encadeada em Array
- ✓ retirados os *defines* para tamanho do vetor, posição inicial e NULO

```
1  #ifndef LISTA_H
2  #define LISTA_H
3
4
5  #define TRUE 1 /*define tipo booleano – não existe em C*/
6  #define FALSE 0
7  #define boolean int /*define um tipo booleano*/
8
9  #define ERRO -32000
10
11 typedef int ITEM;
12
13 typedef struct lista_ LISTA;
14
...
```

TAD Lista Não Ordenada – Exemplo de Definição da Interface

Arquivo Lista.h

```
...
17 LISTA* lista_criar(void);
18 boolean lista_apagar(LISTA **lista); /*ALTERAÇÃO*/
19 boolean lista_inserir(LISTA *lista, ITEM item);
20 boolean lista_inserir_pos(LISTA *lista, int pos, ITEM item);
21 boolean lista_remover(LISTA *lista, int chave);
22 boolean lista_remover_pos(LISTA *lista, int pos);
23 int lista_busca(int chave, LISTA *lista);
24 int lista_tamanho(LISTA *lista);
25 boolean lista_vazia(LISTA *lista);
26 boolean lista_cheia(LISTA *lista);
27 void lista_imprimir(LISTA *lista);
28
29 #endif
```

TAD Lista Não Ordenada – Exemplo de Implementação Encadeada Dinâmica

Arquivo Lista.c

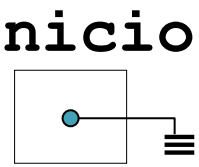
- ✓ alteração nas estruturas que armazenam a lista
- ✓ nova implementação das funções

```
1 #include "lista.h"
2
3 typedef struct no_ NO;
4
5 struct no_{
6     ITEM item;
7     NO *proximo;
8 };
9
10 struct lista_{
11     NO *inicio;
12 };
```

TAD Lista Não Ordenada – Exemplo de Implementação Encadeada Dinâmica

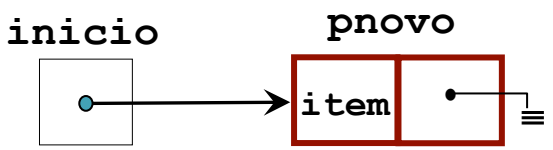
Arquivo Lista.c

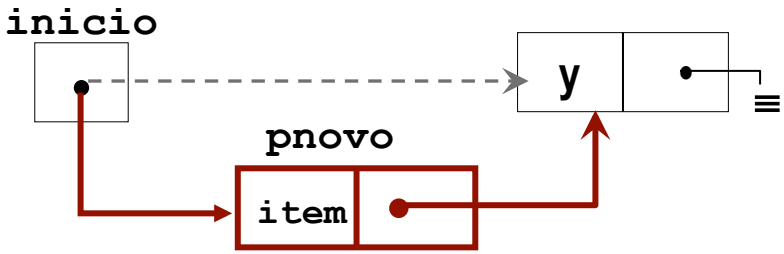
```
1 /*Criação e inicialização da lista*/
2 LISTA* lista_criar(void){
3   LISTA* lista = (LISTA *) malloc(sizeof(LISTA));
4   if(lista != NULL) {
5      lista->inicio = NULL;
6  }
7  return (lista);
8 }
```



Arquivo Lista.c

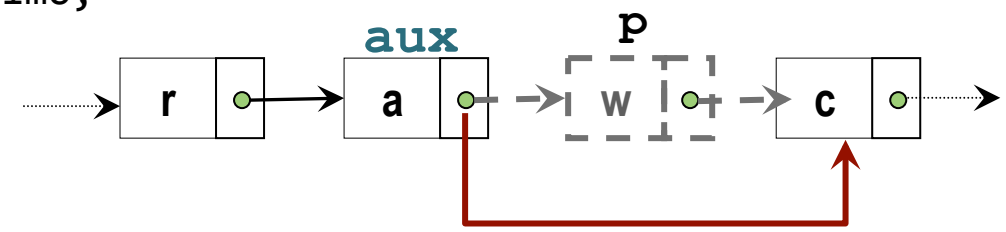
```
1 /*Insere um novo nó no inicio da lista. PARA LISTAS NÃO ORDENADAS*/
2 boolean lista_inserir(LISTA *lista, ITEM item){
3
4    if ((!lista_cheia(lista)) && (lista != NULL)) {
5       NO *pnovo = (NO *) malloc(sizeof (NO));
6       if (lista->inicio == NULL){ /*se lista vazia*/
7          pnovo->item = item;
8          pnovo->proximo = NULL;
9          lista->inicio = pnovo;
10        }
11        else {
12           pnovo->item = item;
13           pnovo->proximo = lista->inicio;
14           lista->inicio = pnovo;
15        }
16
17 return (TRUE);
18     } else
19       return (FALSE);
20 }
```





Arquivo Lista.c

```
 /*Remove um elemento da lista*/
1 boolean lista_remover(LISTA *lista, int chave) {
2   if (lista != NULL){
3     NO *p = lista->inicio;
4     NO *aux = NULL;
5      while(p != NULL && (p->item) != chave) { /*procura chave ou fim lista*/
6         aux = p;  /*aux - guarda posição anterior ao nó sendo pesquisado (p)*/
7         p = p->proximo;
8      }
9      if(p != NULL) {
10          if(p == lista->inicio) { /*se a chave está no 1o nó*/
11             lista->inicio = p->proximo;
12             p->proximo = NULL;
13          }
14          else {
15             aux->proximo = p->proximo;
16             p->proximo = NULL;
17          }
18       free(p);
19       return (TRUE);
20    }
21  } return (FALSE);
22 }
```



Arquivo Lista.c

```
 /*Apaga a lista*/
1 void lista_apagar(LISTA **ptr){
2      if (*ptr == NULL)
3          return;
4
5    /*... Código para percorrer a lista toda e
6  desalocar nó por nó ... */
7
8      free(*ptr);
9      *ptr = NULL;
10 }
```

Listas Encadeadas Dinâmicas

- Pontos Fortes
  - não é necessário pré-definir um tamanho máximo para a lista
    - número de elementos da lista fica limitado apenas à quantidade de memória principal disponível
    - não há alocação desnecessária de espaço
  - acessar o dado apontado por um ponteiro **`p`** não requer cálculo de endereço, como acontece com **`vetor[i]`**
- Desvantagens
  - acesso não é indexado
    - necessário percorrer **`i`** nós para encontrar o **`i`**-ésimo elemento
  - consumo de tempo para alocação e liberação de memória em operações de inserção e remoção

Exercício –
PARA ENTREGA EM 25/09

- Exercício3_Lista.pdf