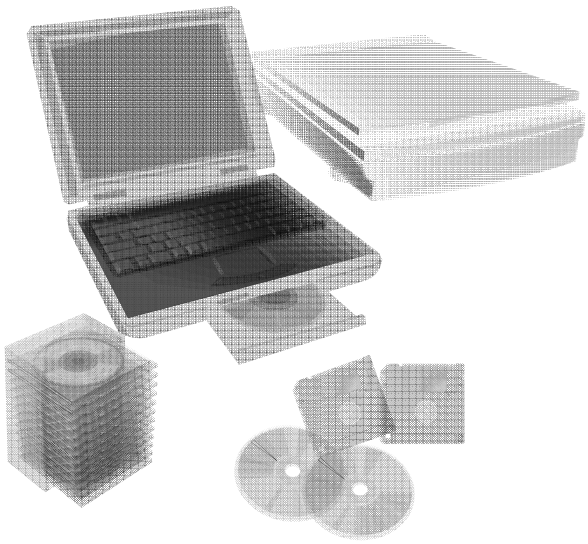
Introdução à Computação II



Listas dinâmicas encadeadas

Profa.: ***Adriane Beatriz de Souza Serapião***

adriane.serapiao@unesp.br

Listas dinâmicas encadeadas

- **Definição: é uma lista encadeada de pares, onde cada par é representado por um registro, constituído por: (elemento, ponteiro).**

- **<u>Desvantagens</u>:**
  - **exige mais espaço (existe um ponteiro adicional por elemento).**
  - **não é possível acessar um elemento diretamente.**
  - **acessar um elemento exige um caminhamento na lista na ordem exibida pelos elementos (leitura linear seguindo ponteiros).**
- **<u>Vantagens</u>:**
  - **Operações de inserção e remoção de elementos.**

Listas dinâmicas encadeadas

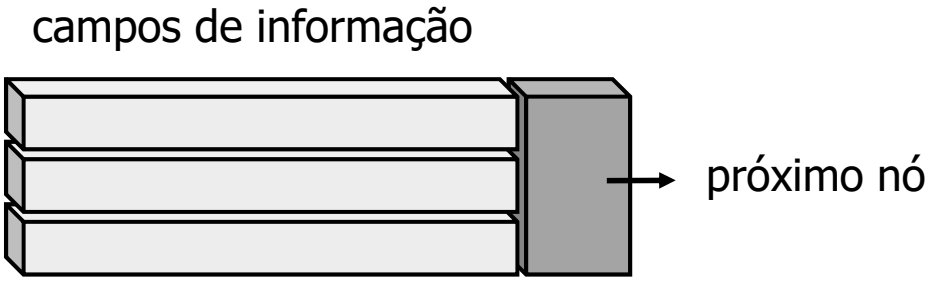
- **Ponteiros**
  - **Estruturas de dados dinâmicas: estruturas de dados que contém ponteiros para si próprias.**

```
struct lista {
  char nome_tarefa[30];
     float duracao;
     char responsavel[30];
  ...
  lista *prox;
};
```

ponteiro para a própria estrutura lista

Listas dinâmicas encadeadas

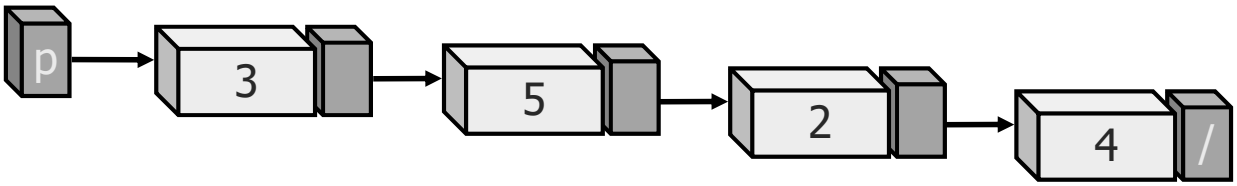
- **Representação gráfica de um elemento da lista:**



  - **Cada item é encadeado com o seguinte, mediante uma variável do tipo ponteiro.**
  - **Permite utilizar posições não contíguas de memória.**
  - **É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.**

Listas dinâmicas encadeadas

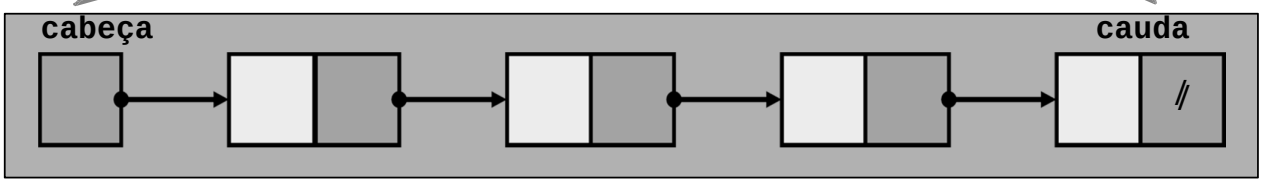
- **Cada item em particular de uma lista pode ser chamado de elemento, nó, célula, ou item.**
- **O apontador para o início da lista também é tratado como se fosse uma célula (cabeça), para simplificar as operações sobre a lista.**
- **O símbolo / representa o ponteiro nulo (NULL), indicando o fim da lista.**



Listas dinâmicas encadeadas

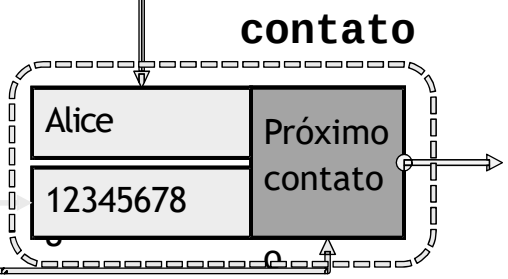
- O primeiro nó da lista é chamado cabeça (*head*).
- Sua função é marcar o início da lista.
- Seu conteúdo é irrelevante.
- O endereço de uma lista encadeada é o endereço do nó cabeça.

- O fim de uma lista encadeada chamado cauda (*tail*).
- Seu conteúdo é um ponteiro nulo (NULL)
- A cauda é indicada pelo símbolo de aterramento ou pelo símbolo "/".



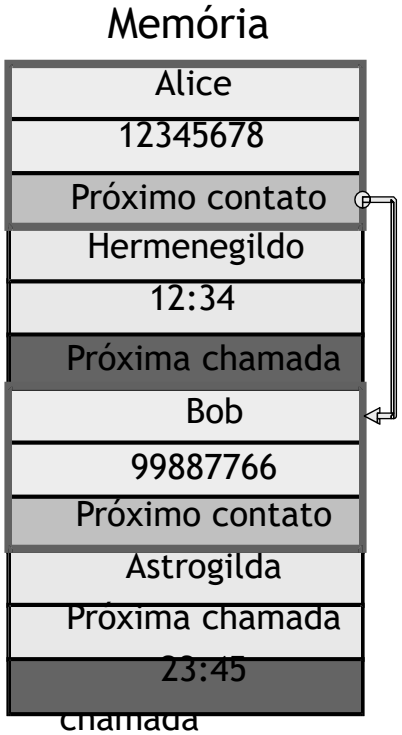
Listas dinâmicas encadeadas

```
struct contato {
   char  nome[20];
   char  número[14]
   struct contato *prox;
};
```



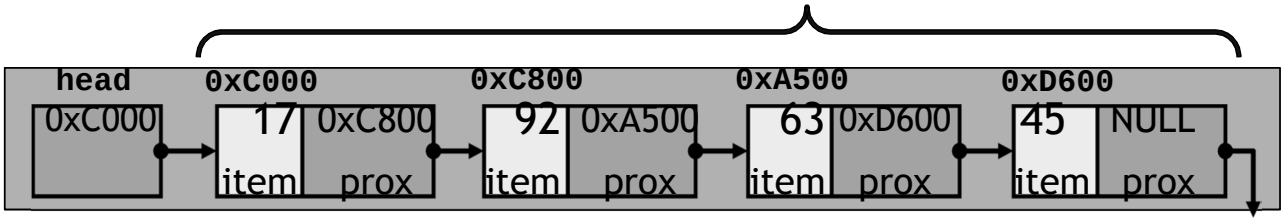
Listas dinâmicas encadeadas

- Os nós que armazenam elementos consecutivos da sequência não ficam necessariamente em posições consecutivas da memória.



Listas dinâmicas encadeadas

Lista Encadeada com 04 nós:



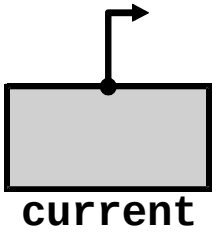

```
struct TipoLista
{  int  item;
   struct TipoLista *prox;
};

TipoLista *head;
```

Nó	Valor
head	0xC000
head->item	17
head->prox	0xC800
head->prox->item	92
head->prox->prox	0xA500
head->prox->prox->item	63

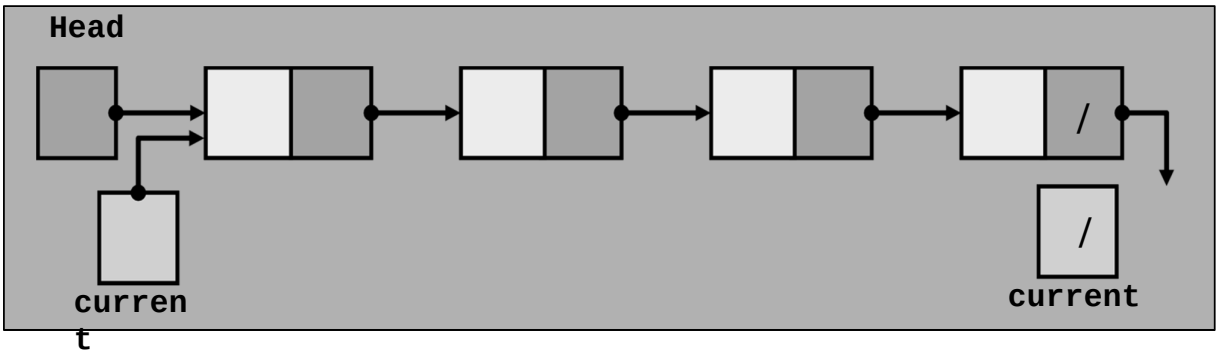
Listas dinâmicas encadeadas

- Em listas longas, o acesso aos nós mais distantes do nó cabeça pode ser tornar confuso, tomando-se o campo **prox** sucessivas vezes.
- Em vez disso, vamos definir uma referência temporária externa chamada **current**, para indicar o nó da lista correntemente observado.

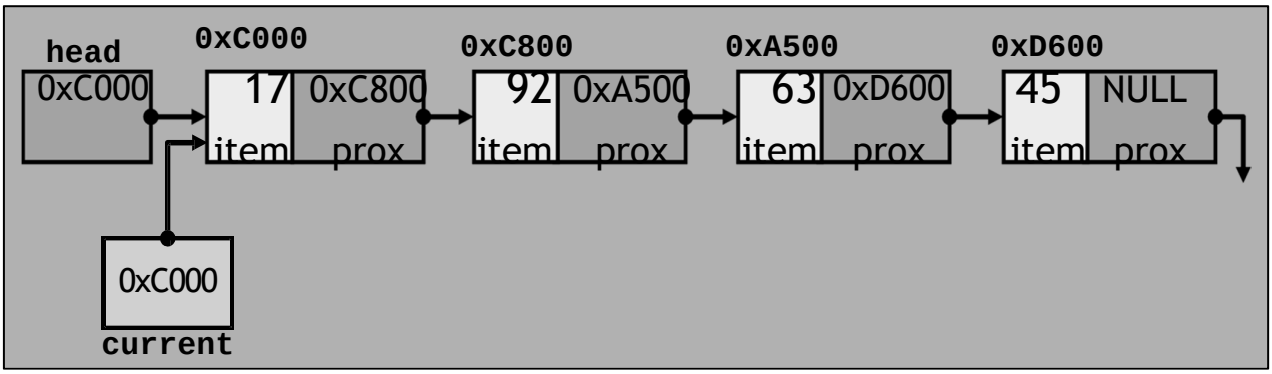


Listas dinâmicas encadeadas

- Algoritmo de travessia:
  1. Inicialmente, **`current`** recebe o endereço da cabeça da lista (**`head`**).
  2. A cada iteração, seu valor é ajustado para o endereço do próximo nó (campo **`prox`**).
  3. O processo termina ao atingir a cauda da lista (ponteiro **`NULL`**).

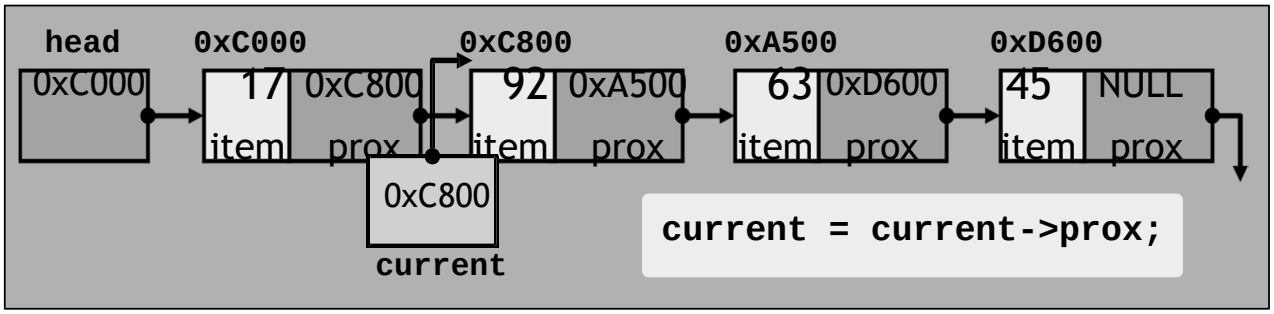


Listas dinâmicas encadeadas



Nó	Valor
`current`	0xC000
`current->item`	17
`current->prox`	0xC800
`current->prox->item`	92

Listas dinâmicas encadeadas



Nó	Valor
`current`	0xC800
`current->item`	92
`current->prox`	0xA500
`current->prox->item`	63

Operações sobre listas dinâmicas encadeadas

- **Podemos realizar algumas operações sobre uma lista encadeadas, tais como:**
  - **Inserir itens;**
  - **Retirar itens;**
  - **Buscar itens.**

- **Para manter a lista ordenada, após realizar alguma dessas operações, será necessário apenas movimentar alguns ponteiros (de um a três elementos).**

Listas dinâmicas encadeadas

Busca

- Partindo da cabeça da lista (*lista), percorre a lista (travessia) até que:
  - Encontre o fim da lista (**p == NULL**); ou
  - Encontre o nó que contenha **x**.

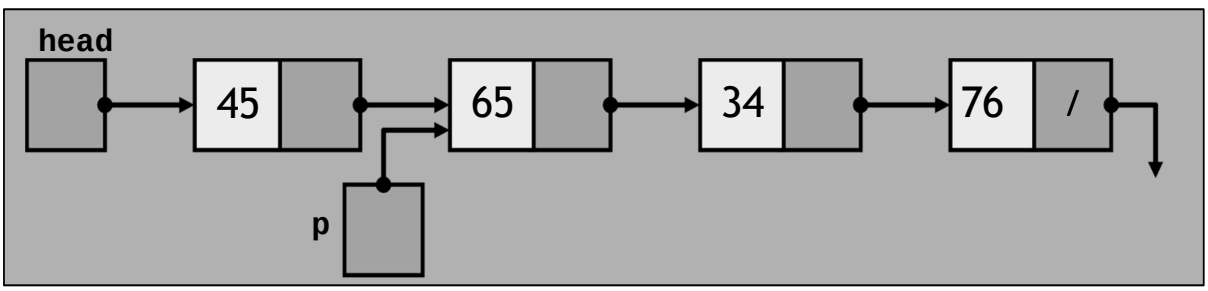
```
struct {
     TipoItem item;
     TipoLista *prox;
} TipoLista;
```

```
TipoLista *busca(TipoLista *lista, TipoItem x)
{
   TipoLista *p;
   p = lista->prox;
   while ((p !=    NULL) &&    (p->item !=  x))
      p = p->prox;
   return p;
}
```

Listas dinâmicas encadeadas

Inserção de itens

- Deseja-se inserir um novo nó de conteúdo **y** entre o nó apontado por **p** e o nó seguinte.
- Exemplo:
  - **p** aponta para o nó **65** e desejamos inserir um novo nó com **item=50** após **p**.

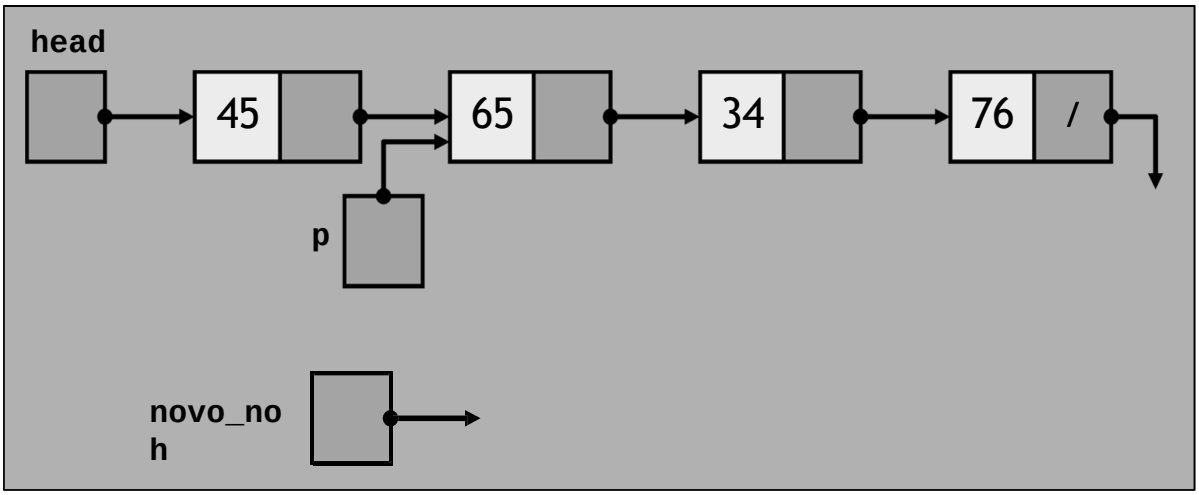


Listas dinâmicas encadeadas

Inserção de itens

1. Criar de ponteiro auxiliar

```
TipoLista *novo_noh;
```

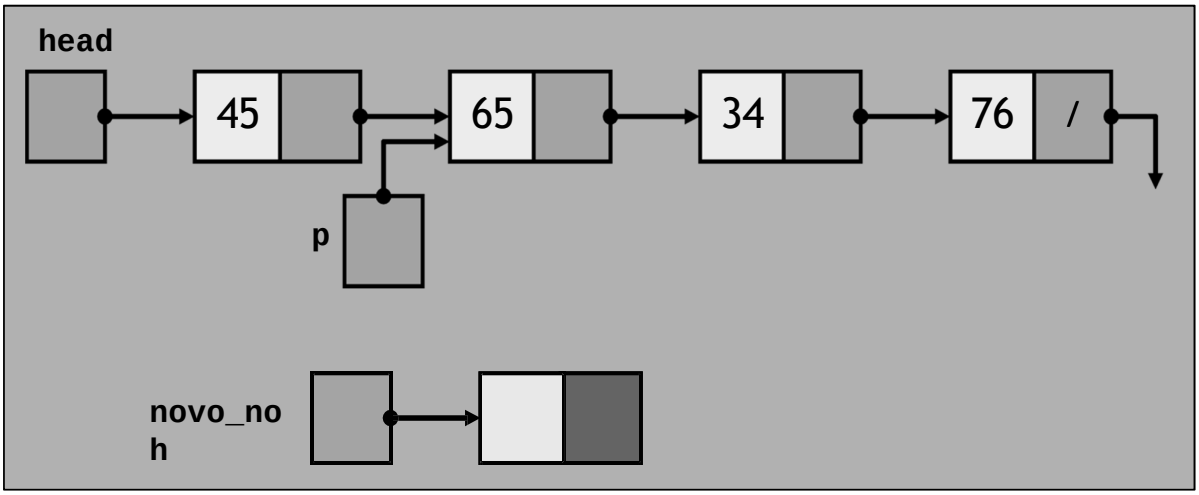


Listas dinâmicas encadeadas

Inserção de itens

2. Alocar espaço para novo nó

```
novo_noh = (TipoLista*) malloc(sizeof(TipoLista));
```

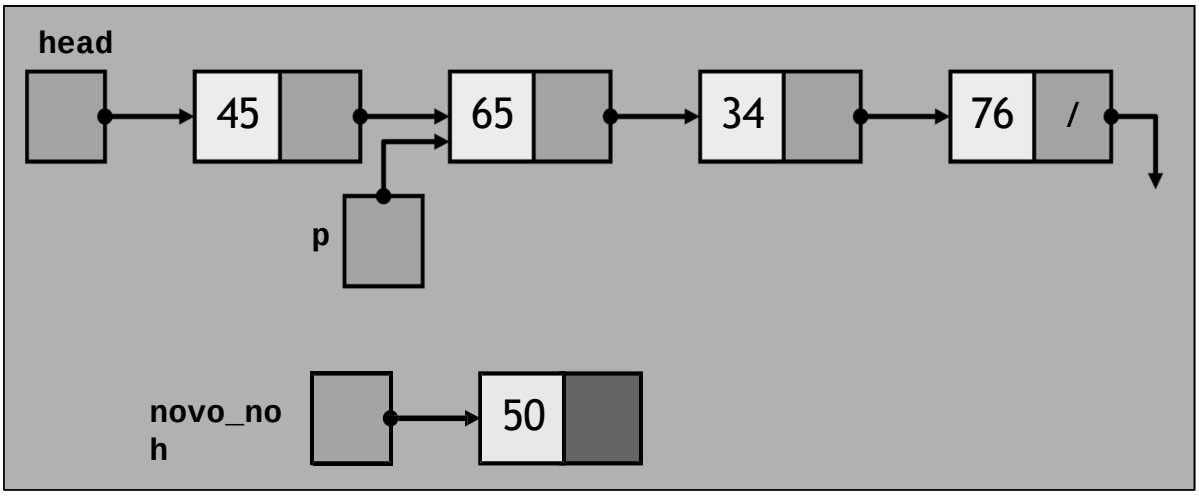


Listas dinâmicas encadeadas

Inserção de itens

3. Preencher conteúdo do novo nó

```
novo_noh->item    =    50;
```

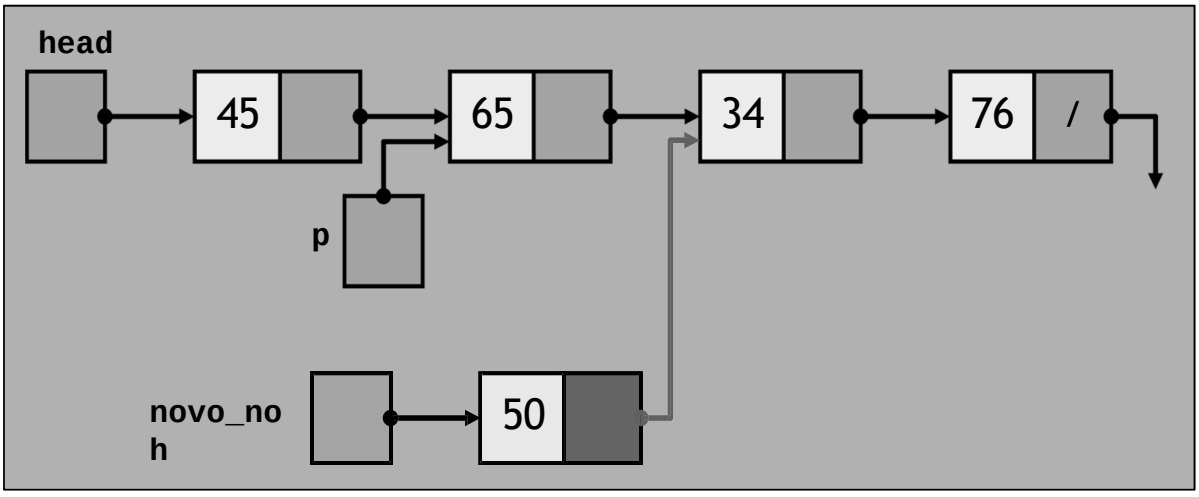


Listas dinâmicas encadeadas

Inserção de itens

4. Conectar novo nó ao nó posterior

```
novo_noh->prox = p->prox;
```

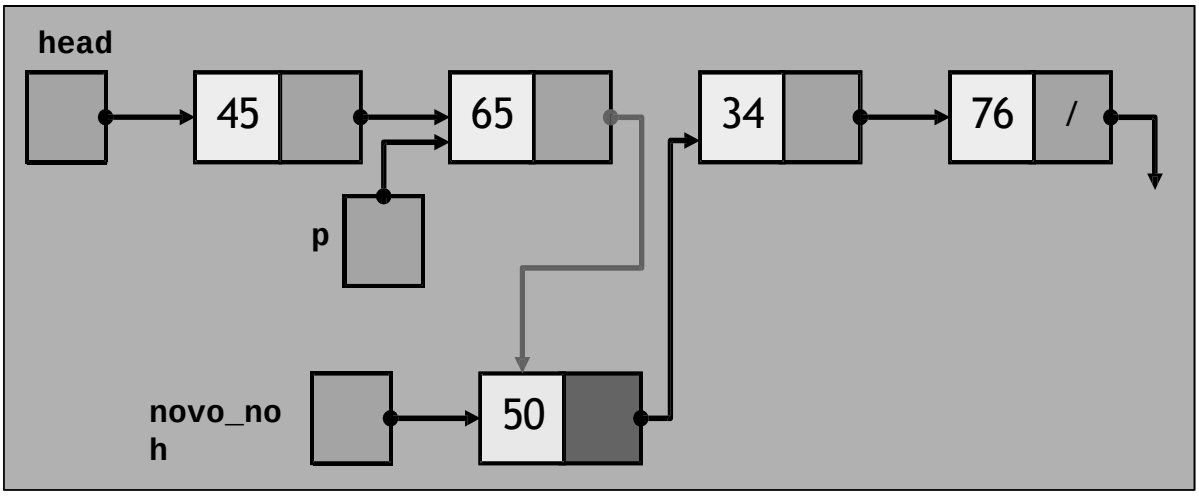


Listas dinâmicas encadeadas

Inserção de itens

5. Conectar nó anterior ao novo nó

```
p->prox = novo_noh;
```



Listas dinâmicas encadeadas

Remoção de itens

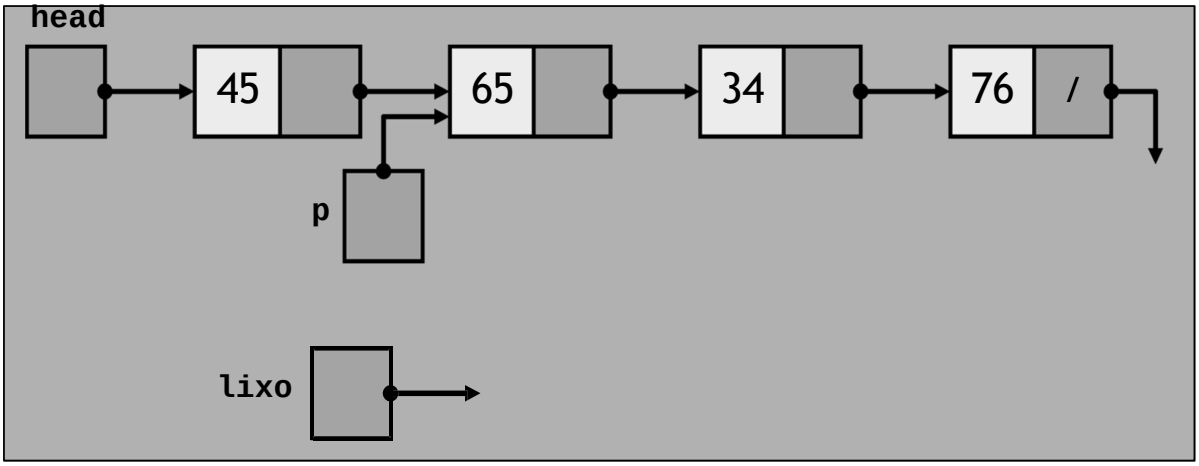
- Deseja-se remover um nó de uma lista.
- Qual nó deve ser especificado?
  - Se for o próprio nó, não será possível alterar os ponteiros dos nós vizinhos para manter encadeamento.
  - É melhor apontar para o nó anterior ao que se deseja remover.
- Exemplo:
  - Remover nó **34** da lista.

Listas dinâmicas encadeadas

Remoção de itens

1. Criar ponteiro auxiliar

```
TipoLista *lixo;
```

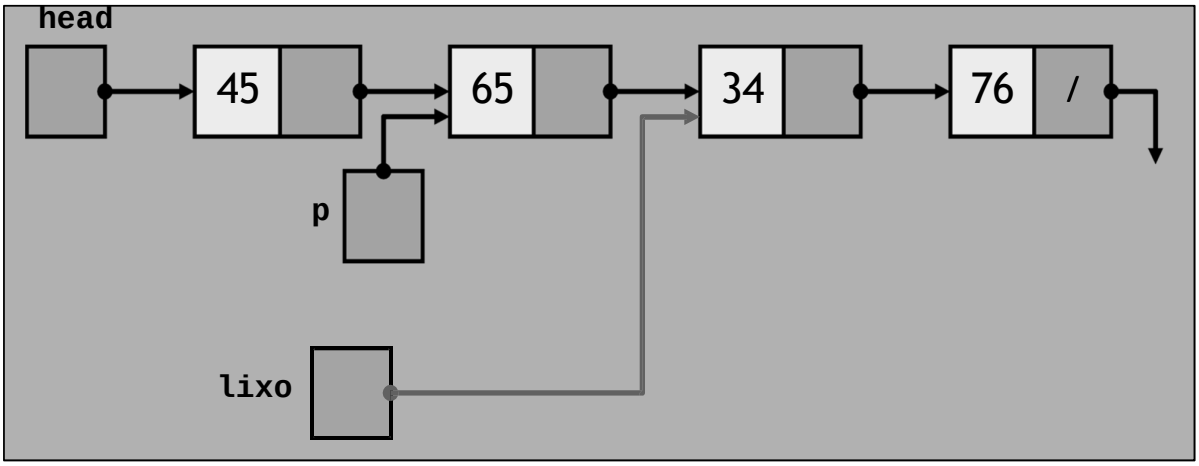


Listas dinâmicas encadeadas

Remoção de itens

2. Conectar ponteiro ao nó a ser removido

```
lixo = p->prox;
```

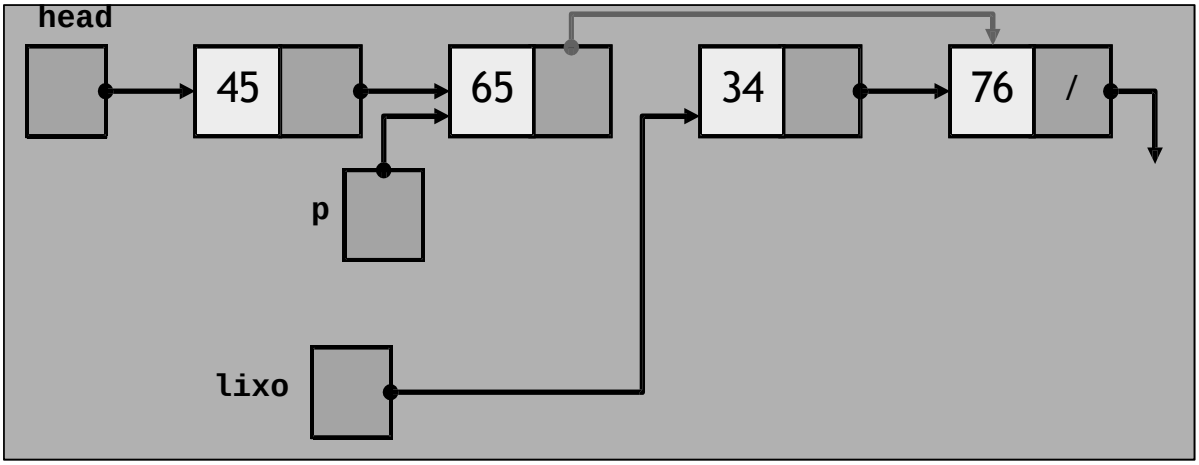


Listas dinâmicas encadeadas

Remoção de itens

3. Conectar nó anterior com o nó posterior

```
p->prox = lixo->prox;
```

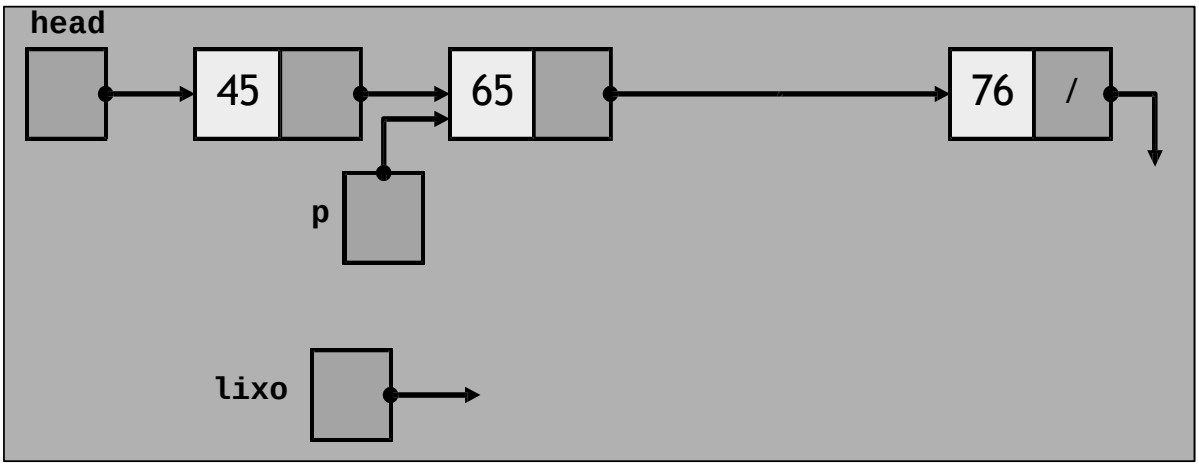


Listas dinâmicas encadeadas

Remoção de itens

4. Liberar espaço de memória do nó removido

```
free(lixo);
```



Listas dinâmicas encadeadas

Não derrube pontes antes de atravessá-las

❖Tenha atenção para alterar na ordem correta os links dos nós de uma lista.

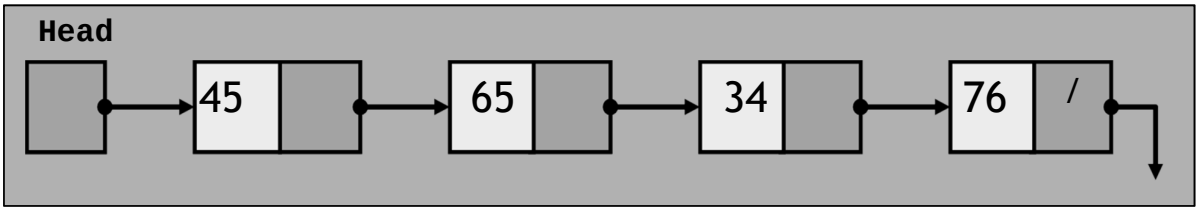
❖Caso contrário, poderá perder acesso a um ou mais nós da lista.

Lista dinâmica encadeada

Busca seguida de Remoção

❖Dado um inteiro x, remover da lista o primeiro nó que contém x.

❖Se tal nó não existir, não é preciso fazer nada.



Lista dinâmica encadeada

Busca seguida de Inserção

- Inserir na lista um novo nó com conteúdo x imediatamente antes do primeiro nó que tiver conteúdo y.
- Se tal nó não existir, inserir x no fim da lista.

Operações sobre listas dinâmicas

- **Outras operações possíveis:**
  - **Criar uma lista**
  - **Destruir uma lista**
  - **Ordenar uma lista**
  - **Intercalar duas listas**
  - **Concatenar duas listas**
  - **Dividir uma lista em duas**
  - **Copiar uma lista em outra**

Operações sobre listas dinâmicas

```
typedef struct {
      TipoItem item;
      TipoLista *prox;
} TipoLista;

void FLVazia (TipoLista *lista)
{
        lista = NULL;
};
```

```
int Vazia (TipoLista *lista)
{
   return (lista == NULL);
};
```


Operações sobre listas dinâmicas

```
void Insere_fim (TipoItem x, TipoLista *lista)
{
   TipoLista *Novo, *p;
   Novo->prox = (TipoLista *) malloc(sizeof(TipoLista));
   Novo->item = x;
   Novo->prox = NULL;
   p = lista->prox;
   while (p->prox != NULL) p = p->prox;
   p->prox = Novo;
};

void Insere_inicio (TipoItem x, TipoLista *lista)
{
   TipoLista *p;
   p->prox = (TipoLista *) malloc(sizeof(TipoLista));
   p->item = x;
   p->prox = lista;
   lista = p;
};
```

Operações sobre listas dinâmicas

```
void Imprime (TipoLista *lista)
{
   TipoLista *aux;
   aux = Lista->prox;
   while (aux != NULL)
   {
       printf("%d\n",aux->item);
       aux = aux->prox;
   };
};
```


Listas dinâmicas – 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 ponteiros.**
- **Quando usar:**
  - **quando for possível fazer uma boa previsão do espaço utilizado (lista principal + lista auxiliar) e quando o ganho dos movimentos sobre a perda do acesso direto a cada elemento for compensador.**

Atenção!

- **Há diversas maneiras de construir a estrutura de uma lista dinâmica encadeada.**
- **Aqui foi apresentada uma lista encadeada simples com cabeça.**

Exercícios – listas

- **Faça um programa para inverter uma lista:**

```
void InverteLista(TipoLista *p)
{
  Apontador q, r, s;
  q = NULL;
  r = p;
  while (r != NULL) {
    s = r->prox;
    r->prox = q;
    q = r;
    r = s
  };
  p = q
};
```

Exercícios – listas

- **Implemente algumas das operações básicas com listas dinâmicas que não foram apresentadas aqui:**
  - **Inserir elemento em uma posição qualquer da lista;**
  - **Inserir elemento no início da lista;**
  - **Inserir elemento no fim da lista;**
  - **Retirar elemento no início da lista;**
  - **Retirar elemento no fim da lista;**
  - **Retirar elemento de uma posição qualquer da lista;**
  - **Buscar um elemento qualquer dentro de uma lista;**
  - **Dada uma posição qualquer em uma lista, acessar seu conteúdo;**
  - **Concatenar listas;**
  - **Copiar uma lista em outra;**
  - **Trocar dois nós de posição.**

Implementações recursivas

- Listas são objetos recursivos:
  - Ao remover o primeiro elemento de uma lista encadeada, resta-nos uma lista encadeada menor.
- Essa percepção leva a transformar processamentos simples de listas em algoritmos eficientes de divisão e conquista.

Operações recursivas sobre listas encadeadas

```
int listaVazia(TipoLista * lista)
{
   return (lista == NULL);
}


void liberaListaRec(TipoLista * lista)
{
   if (!listaVazia(lista))
   {
      liberaListaRec(lista->prox);
      free(lista);
   }
}
```


Operações recursivas sobre listas encadeadas

```
void imprimeListaInvertidaRec(TipoLista *lista)
{
   if (!listaVazia(lista))
   {
      imprimeListaInvertidaRec(lst->prox);
      printf("info: %d\n",lista->item);
   }
}


void imprimeListaRec(TipoLista *lista)
{
   if ( !listaVazia(lista))
   {
       printf("info: %d\n",lista->item);
      imprimeListaRec(lista->prox);
    }
}
```

Operações recursivas sobre listas encadeadas

```
int listaIgualRec(TipoLista *lista1, TipoLista * lista2)
{
   if (lista1 == NULL && lista2 == NULL)
      return 1;
   else if (lista1 == NULL || lista2 == NULL)
      return 0;
   else
   return    (lista1->item    ==    lista2->item)    &&
   listaIgualRec(lista1->prox,lista2->prox);
}
```


Operações recursivas sobre listas encadeadas

```
TipoLista * retiraListaRec(TipoLista *lista, TipoItem a)
{
   TipoLista *t;
   if (!listaVazia(lista))
   {
       if (lista->item == a)
       {
           t = lista;
           lista = lista->prox;
          free(t);
       }
       else
       {
           lista->prox= retiraListaRec (lista->prox,a);
        }
    }
   return lista;
}
```

Operações recursivas sobre listas encadeadas

```
TipoLista *insereListaOrdenadaRec(TipoLista *lista, TipoItem a)
{
   TipoLista *novo;
   TipoLista *ant = NULL;
   TipoLista *p = lista;
   while (p != NULL && p->item < a)
   {
        ant = p;
        p = p->prox;
   }
   novo = (TipoLista *) malloc(sizeof(TipoLista));
   novo->item = a;
   if (ant == NULL) {
      novo->prox = lista;
      lista = novo;
   }
   else {
       novo->prox = ant->prox;
       ant->prox = novo;
    }
   return lista;
}
```


Bibliografia

- **ZIVIANI, N. *Projeto de Algoritmos com Implementações em Pascal e C*, (4a. ed.). Thomson, 2011.**