

# Listas lineares

## Aula 16

Diego Padilha Rubert

Faculdade de Computação  
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

# Conteúdo da aula

- 1 Introdução
- 2 Definição
- 3 Operações sobre listas lineares com cabeça
- 4 Operações sobre listas lineares sem cabeça
- 5 Exercícios

- ▶ as próximas estruturas de dados que aprendemos depois de listas de prioridades
- ▶ diversas aplicações importantes para organização de informações na memória tais como representações alternativas para expressões aritméticas, armazenamento de argumentos de funções, compartilhamento de espaço de memória

- ▶ as próximas estruturas de dados que aprendemos depois de listas de prioridades
- ▶ diversas aplicações importantes para organização de informações na memória tais como representações alternativas para expressões aritméticas, armazenamento de argumentos de funções, compartilhamento de espaço de memória

# Definição

- ▶ uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
- ▶ relação se expressa apenas pela ordem relativa entre os elementos
- ▶ nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- ▶ cada informação contida na lista é um registro contendo os dados relacionados, chamados de célula
- ▶ usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- ▶ dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros

# Definição

- ▶ uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
- ▶ relação se expressa apenas pela ordem relativa entre os elementos
- ▶ nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- ▶ cada informação contida na lista é um registro contendo os dados relacionados, chamados de célula
- ▶ usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- ▶ dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros

# Definição

- ▶ uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
- ▶ relação se expressa apenas pela ordem relativa entre os elementos
- ▶ nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- ▶ cada informação contida na lista é um registro contendo os dados relacionados, chamados de célula
- ▶ usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- ▶ dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros

# Definição

- ▶ uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
- ▶ relação se expressa apenas pela ordem relativa entre os elementos
- ▶ nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- ▶ cada informação contida na lista é um registro contendo os dados relacionados, chamados de célula
- ▶ usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- ▶ dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros



# Definição

- ▶ uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
- ▶ relação se expressa apenas pela ordem relativa entre os elementos
- ▶ nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- ▶ cada informação contida na lista é um registro contendo os dados relacionados, chamados de célula
- ▶ usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- ▶ dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros

# Definição

- ▶ uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
- ▶ relação se expressa apenas pela ordem relativa entre os elementos
- ▶ nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- ▶ cada informação contida na lista é um registro contendo os dados relacionados, chamados de célula
- ▶ usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- ▶ dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros

- ▶ uma **lista linear** é um conjunto de  $n \geq 0$  células  $c_1, c_2, \dots, c_n$  determinada pela ordem relativa desses elementos:
  - (i) se  $n > 0$  então  $c_1$  é a primeira célula;
  - (ii) a célula  $c_i$  é precedida pela célula  $c_{i-1}$ , para todo  $i$ ,  $1 < i \leq n$ .
- ▶ as operações básicas sobre uma lista linear são as seguintes:
  - ▶ busca;
  - ▶ inclusão; e
  - ▶ remoção.
- ▶ dependendo da aplicação, muitas outras operações também podem ser realizadas sobre essa estrutura

- ▶ uma **lista linear** é um conjunto de  $n \geq 0$  células  $c_1, c_2, \dots, c_n$  determinada pela ordem relativa desses elementos:
  - (i) se  $n > 0$  então  $c_1$  é a primeira célula;
  - (ii) a célula  $c_i$  é precedida pela célula  $c_{i-1}$ , para todo  $i$ ,  $1 < i \leq n$ .
- ▶ as operações básicas sobre uma lista linear são as seguintes:
  - ▶ busca;
  - ▶ inclusão; e
  - ▶ remoção.
- ▶ dependendo da aplicação, muitas outras operações também podem ser realizadas sobre essa estrutura

- ▶ uma **lista linear** é um conjunto de  $n \geq 0$  células  $c_1, c_2, \dots, c_n$  determinada pela ordem relativa desses elementos:
  - (i) se  $n > 0$  então  $c_1$  é a primeira célula;
  - (ii) a célula  $c_i$  é precedida pela célula  $c_{i-1}$ , para todo  $i$ ,  $1 < i \leq n$ .
- ▶ as operações básicas sobre uma lista linear são as seguintes:
  - ▶ busca;
  - ▶ inclusão; e
  - ▶ remoção.
- ▶ dependendo da aplicação, muitas outras operações também podem ser realizadas sobre essa estrutura

- ▶ listas lineares podem ser armazenadas na memória de duas maneiras distintas:
  - ▶ **alocação estática ou sequencial**: os elementos são armazenados em posições consecutivas de memória, com uso de vetores;
  - ▶ **alocação dinâmica ou encadeada**: os elementos podem ser armazenados em posições não consecutivas de memória, com uso de ponteiros
- ▶ o problema que queremos resolver é que define o tipo de armazenamento a ser usado, dependendo das operações sobre a lista, do número de listas envolvidas e das características particulares das listas
- ▶ já vimos as operações básicas sobre uma lista linear em alocação sequencial

- ▶ listas lineares podem ser armazenadas na memória de duas maneiras distintas:
  - ▶ **alocação estática ou sequencial**: os elementos são armazenados em posições consecutivas de memória, com uso de vetores;
  - ▶ **alocação dinâmica ou encadeada**: os elementos podem ser armazenados em posições não consecutivas de memória, com uso de ponteiros
- ▶ o problema que queremos resolver é que define o tipo de armazenamento a ser usado, dependendo das operações sobre a lista, do número de listas envolvidas e das características particulares das listas
- ▶ já vimos as operações básicas sobre uma lista linear em alocação sequencial

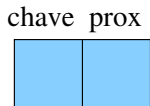
- ▶ listas lineares podem ser armazenadas na memória de duas maneiras distintas:
  - ▶ **alocação estática ou sequencial**: os elementos são armazenados em posições consecutivas de memória, com uso de vetores;
  - ▶ **alocação dinâmica ou encadeada**: os elementos podem ser armazenados em posições não consecutivas de memória, com uso de ponteiros
- ▶ o problema que queremos resolver é que define o tipo de armazenamento a ser usado, dependendo das operações sobre a lista, do número de listas envolvidas e das características particulares das listas
- ▶ já vimos as operações básicas sobre uma lista linear em alocação sequencial



- ▶ listas lineares podem ser armazenadas na memória de duas maneiras distintas:
  - ▶ **alocação estática ou sequencial**: os elementos são armazenados em posições consecutivas de memória, com uso de vetores;
  - ▶ **alocação dinâmica ou encadeada**: os elementos podem ser armazenados em posições não consecutivas de memória, com uso de ponteiros
- ▶ o problema que queremos resolver é que define o tipo de armazenamento a ser usado, dependendo das operações sobre a lista, do número de listas envolvidas e das características particulares das listas
- ▶ já vimos as operações básicas sobre uma lista linear em alocação sequencial

# Definição

- ▶ as células de uma lista linear em alocação encadeada encontram-se dispostas em posições aleatórias da memória e são ligadas por ponteiros que indicam a posição da próxima célula da lista
- ▶ um campo é acrescentado a cada célula da lista indicando o endereço do próximo elemento da lista



# Definição

- ▶ definição de uma célula de uma lista linear encadeada:

```
struct cel {  
    int chave;  
    struct cel *prox;  
};
```

- ▶ definir um novo tipo de dados para as células de uma lista linear em alocação encadeada:

```
typedef struct cel celula;
```

- ▶ uma célula `c` e um ponteiro `p` para uma célula podem ser declarados da seguinte forma:

```
celula c;  
celula *p;
```

# Definição

- ▶ definição de uma célula de uma lista linear encadeada:

```
struct cel {  
    int chave;  
    struct cel *prox;  
};
```

- ▶ definir um novo tipo de dados para as células de uma lista linear em alocação encadeada:

```
typedef struct cel celula;
```

- ▶ uma célula `c` e um ponteiro `p` para uma célula podem ser declarados da seguinte forma:

```
celula c;  
celula *p;
```

- ▶ definição de uma célula de uma lista linear encadeada:

```
struct cel {  
    int chave;  
    struct cel *prox;  
};
```

- ▶ definir um novo tipo de dados para as células de uma lista linear em alocação encadeada:

```
typedef struct cel celula;
```

- ▶ uma célula **c** e um ponteiro **p** para uma célula podem ser declarados da seguinte forma:

```
celula c;  
celula *p;
```

# Definição

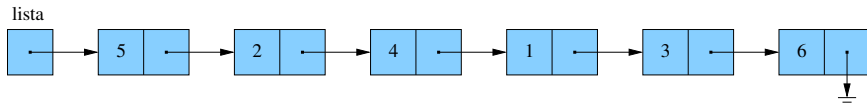
- ▶ se **c** é uma célula então **c.chave** é o conteúdo da célula e **c.prox** é o endereço da célula seguinte
- ▶ se **p** é o endereço de uma célula então **p->chave** é o conteúdo da célula apontada por **p** e **p->prox** é o endereço da célula seguinte
- ▶ Se **p** é o endereço da última célula da lista então **p->prox** vale **NULL**

- ▶ se **c** é uma célula então **c.chave** é o conteúdo da célula e **c.prox** é o endereço da célula seguinte
- ▶ se **p** é o endereço de uma célula então **p->chave** é o conteúdo da célula apontada por **p** e **p->prox** é o endereço da célula seguinte
- ▶ Se **p** é o endereço da última célula da lista então **p->prox** vale **NULL**

- ▶ se **c** é uma célula então **c.chave** é o conteúdo da célula e **c.prox** é o endereço da célula seguinte
- ▶ se **p** é o endereço de uma célula então **p->chave** é o conteúdo da célula apontada por **p** e **p->prox** é o endereço da célula seguinte
- ▶ Se **p** é o endereço da última célula da lista então **p->prox** vale **NULL**



# Definição



- ▶ o endereço de uma lista encadeada é o endereço de sua primeira célula
- ▶ se  $p$  é o endereço de uma lista, podemos dizer que “ $p$  é uma lista” ou ainda “considere a lista  $p$ ”
- ▶ quando dizemos “ $p$  é uma lista”, queremos dizer que “ $p$  é o endereço da primeira célula de uma lista”

- ▶ o endereço de uma lista encadeada é o endereço de sua primeira célula
- ▶ se **p** é o endereço de uma lista, podemos dizer que “**p** é uma lista” ou ainda “considere a lista **p**”
- ▶ quando dizemos “**p** é uma lista”, queremos dizer que “**p** é o endereço da primeira célula de uma lista”

- ▶ o endereço de uma lista encadeada é o endereço de sua primeira célula
- ▶ se **p** é o endereço de uma lista, podemos dizer que “**p** é uma lista” ou ainda “considere a lista **p**”
- ▶ quando dizemos “**p** é uma lista”, queremos dizer que “**p** é o endereço da primeira célula de uma lista”

- ▶ uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa
- ▶ em uma lista linear **com cabeça**, a primeira célula serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante; a primeira célula é a **cabeça** da lista
- ▶ em uma lista linear **sem cabeça** o conteúdo da primeira célula é tão relevante quanto o das demais

- ▶ uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa
- ▶ em uma lista linear **com cabeça**, a primeira célula serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante; a primeira célula é a **cabeça** da lista
- ▶ em uma lista linear **sem cabeça** o conteúdo da primeira célula é tão relevante quanto o das demais

- ▶ uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa
- ▶ em uma lista linear **com cabeça**, a primeira célula serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante; a primeira célula é a **cabeça** da lista
- ▶ em uma lista linear **sem cabeça** o conteúdo da primeira célula é tão relevante quanto o das demais

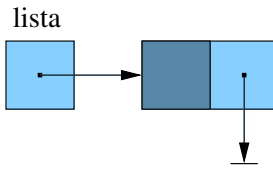
# Definição

- ▶ uma lista linear está vazia se não tem célula alguma
- ▶ para criar uma lista vazia **lista** com cabeça, basta escrever as seguintes sentenças:

```
celula c, *lista;  
c.prox = NULL;  
lista = &c;
```

ou ainda

```
celula *lista;  
lista = (celula *) malloc(sizeof (celula));  
lista->prox = NULL;
```



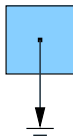


# Definição

- ▶ para criar uma lista vazia **lista** sem cabeça, basta escrever as seguintes sentenças:

```
celula *lista;  
lista = NULL;
```

lista



- ▶ listas lineares com cabeça são mais fáceis de manipular do que aquelas sem cabeça. No entanto, as listas com cabeça têm sempre a desvantagem de manter uma célula a mais na memória

# Definição

- ▶ para imprimir o conteúdo de todas as células de uma lista linear podemos usar a seguinte função:

```
void imprime_lista(celula *lst)
{
    celula *p;

    for (p = lst; p != NULL; p = p->prox)
        printf("%d\n", p->chave);
}
```

- ▶ se `lista` é uma lista linear com cabeça, a chamada da função deve ser:

```
imprime_lista(lista->prox);
```

- ▶ se `lista` é uma lista linear sem cabeça, a chamada da função deve ser:

```
imprime_lista(lista);
```

# Definição

- ▶ para imprimir o conteúdo de todas as células de uma lista linear podemos usar a seguinte função:

```
void imprime_lista(celula *lst)
{
    celula *p;

    for (p = lst; p != NULL; p = p->prox)
        printf("%d\n", p->chave);
}
```

- ▶ se **lista** é uma lista linear com cabeça, a chamada da função deve ser:

```
imprime_lista(lista->prox);
```

- ▶ se **lista** é uma lista linear sem cabeça, a chamada da função deve ser:

```
imprime_lista(lista);
```

# Definição

- ▶ para imprimir o conteúdo de todas as células de uma lista linear podemos usar a seguinte função:

```
void imprime_lista(celula *lst)
{
    celula *p;

    for (p = lst; p != NULL; p = p->prox)
        printf("%d\n", p->chave);
}
```

- ▶ se **lista** é uma lista linear com cabeça, a chamada da função deve ser:

```
imprime_lista(lista->prox);
```

- ▶ se **lista** é uma lista linear sem cabeça, a chamada da função deve ser:

```
imprime_lista(lista);
```

# Operações sobre listas lineares com cabeça

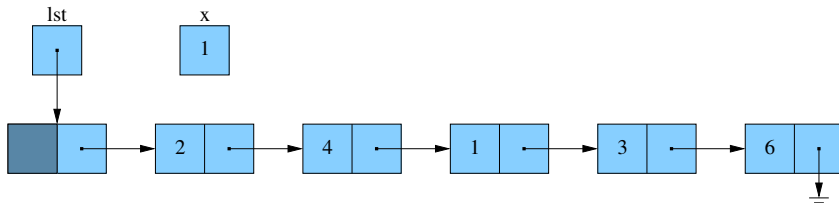
## ► busca não-recursiva

```
celula *busca_C(int x, celula *lst)
{
    celula *p;

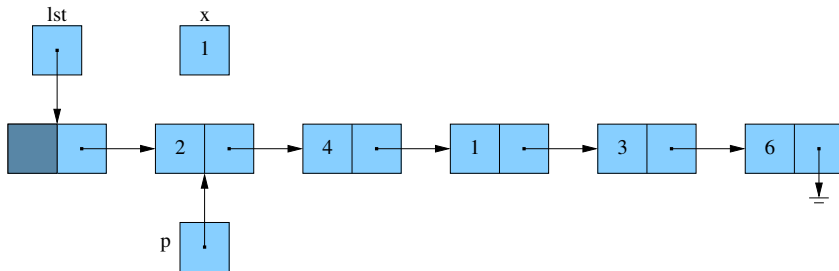
    p = lst->prox;
    while (p != NULL && p->chave != x)
        p = p->prox;

    return p;
}
```

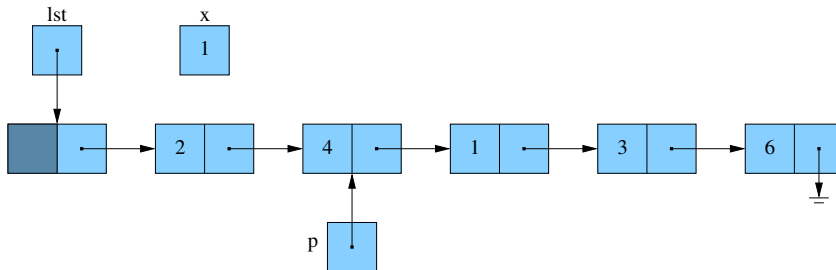
# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça

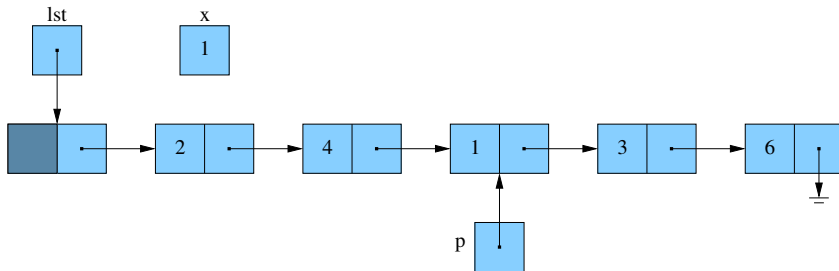


# Operações sobre listas lineares com cabeça





# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça

## ► busca recursiva

```
celula *buscaR_C(int x, celula *lst)
{
    if (lst->prox == NULL)
        return NULL;

    if (lst->prox->chave == x)
        return lst->prox;

    return buscaR_C(x, lst->prox);
}
```

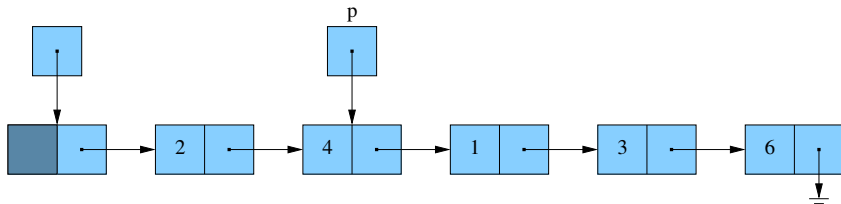
# Operações sobre listas lineares com cabeça

## ► remoção

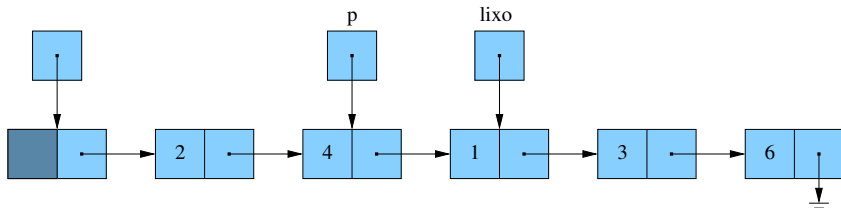
```
void remove_C(celula *p)
{
    celula *lixo;

    lixo = p->prox;
    p->prox = lixo->prox;
    free(lixo);
}
```

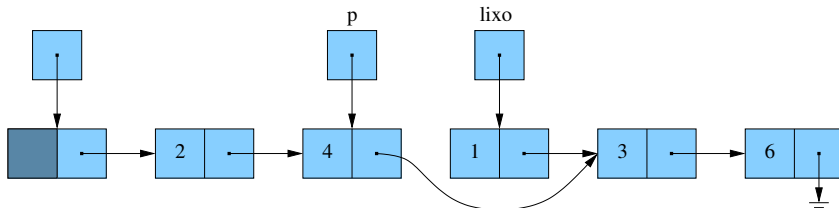
# Operações sobre listas lineares com cabeça



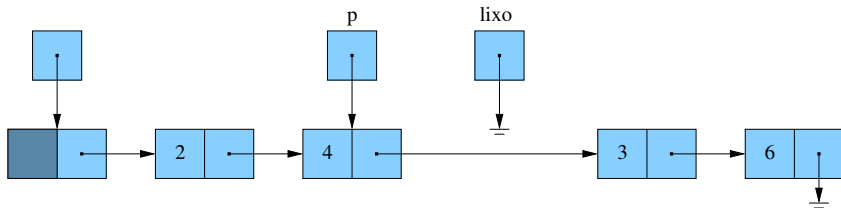
# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça

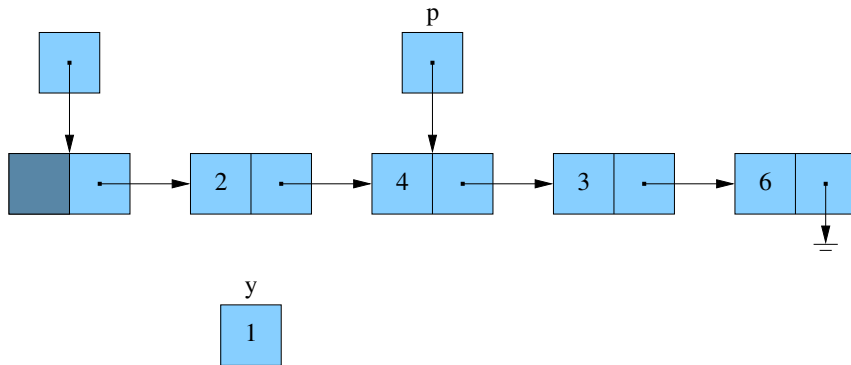
## ► inserção

```
void insere_C(int y, celula *p)
{
    celula *nova;

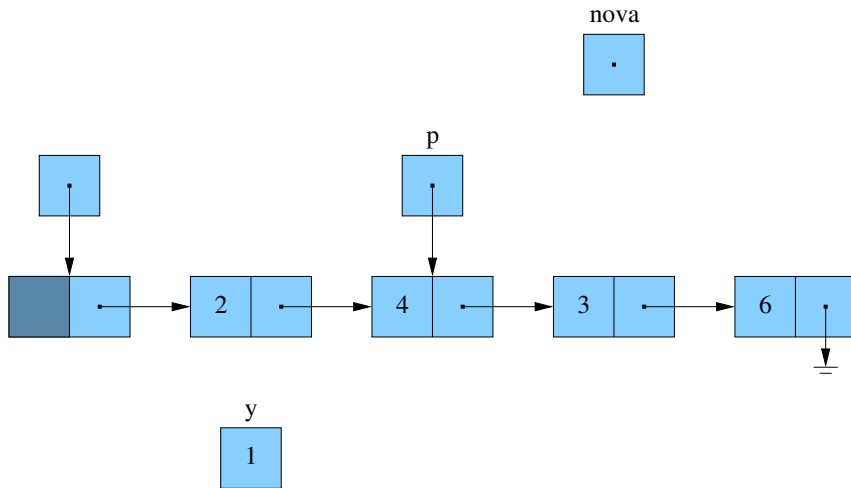
    nova = (celula *) malloc(sizeof (celula));
    nova->chave = y;
    nova->prox = p->prox;
    p->prox = nova;
}
```



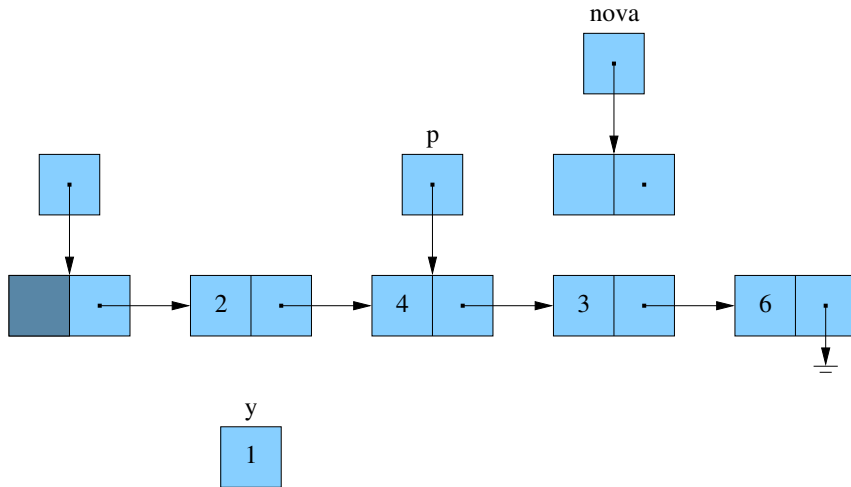
# Operações sobre listas lineares com cabeça



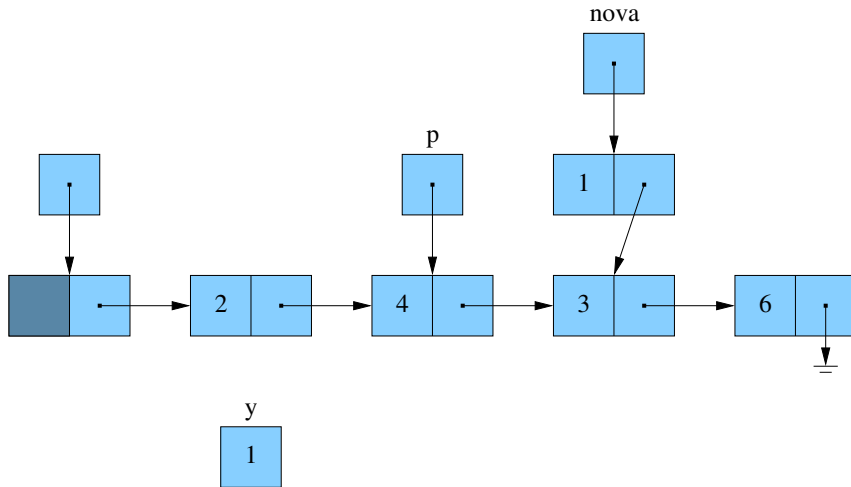
# Operações sobre listas lineares com cabeça



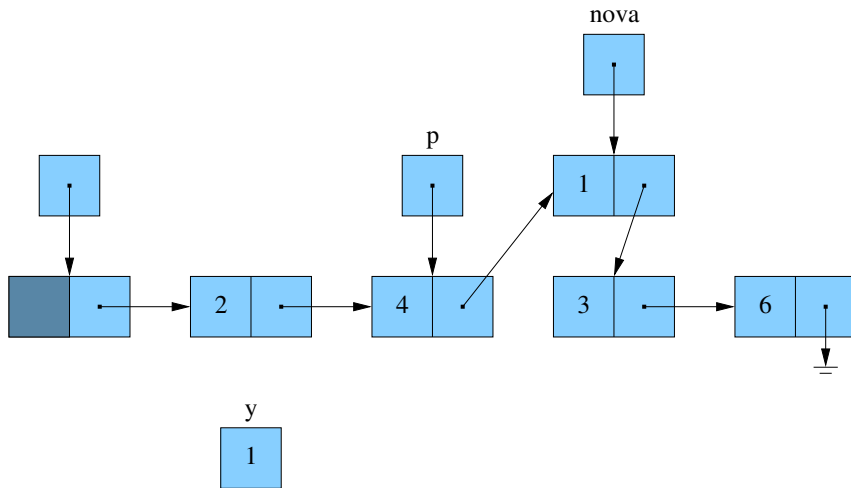
# Operações sobre listas lineares com cabeça



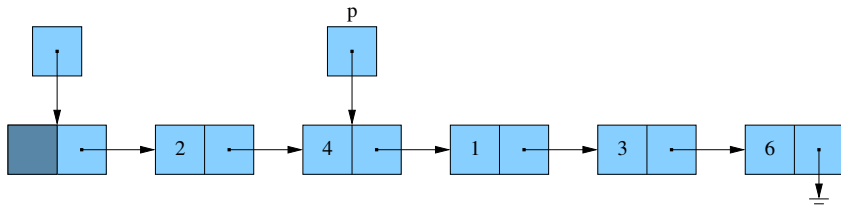
# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



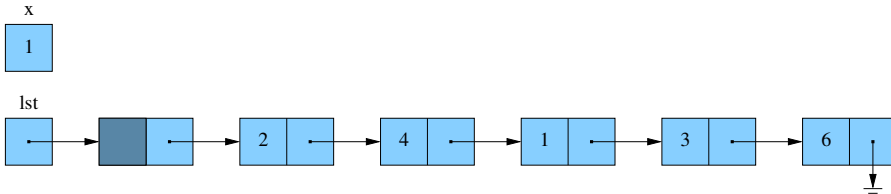
# Operações sobre listas lineares com cabeça

## ► busca seguida de remoção

```
void busca_remove_C(int x, celula *lst)
{
    celula *p, *q;

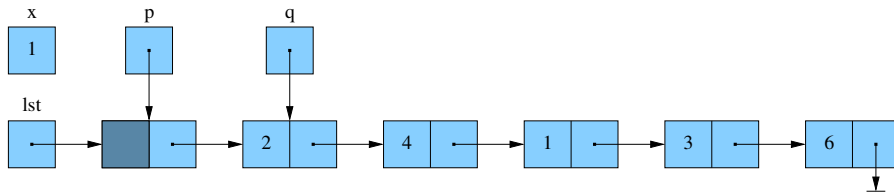
    p = lst;
    q = lst->prox;
    while (q != NULL && q->chave != x) {
        p = q;
        q = q->prox;
    }
    if (q != NULL) {
        p->prox = q->prox;
        free(q);
    }
}
```

# Operações sobre listas lineares com cabeça

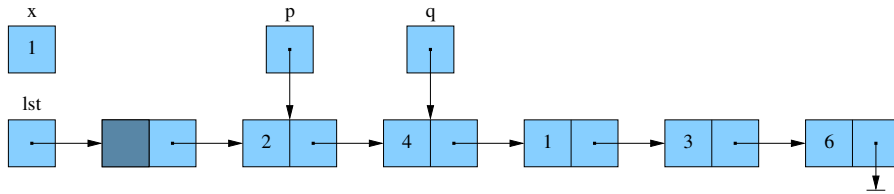




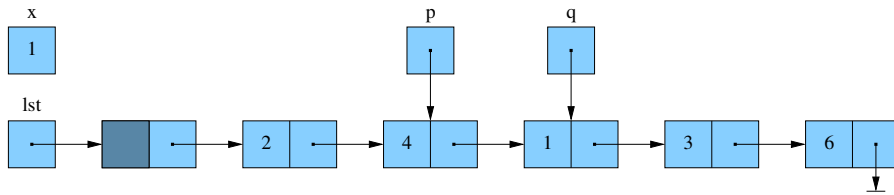
# Operações sobre listas lineares com cabeça



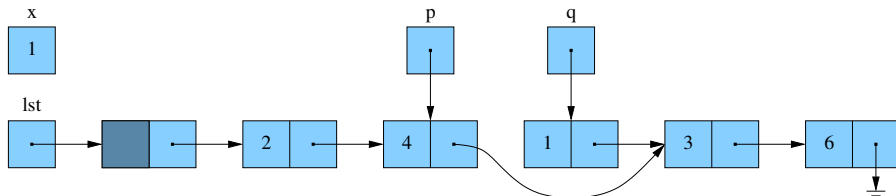
# Operações sobre listas lineares com cabeça



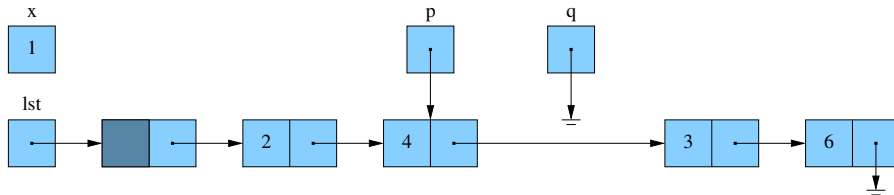
# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



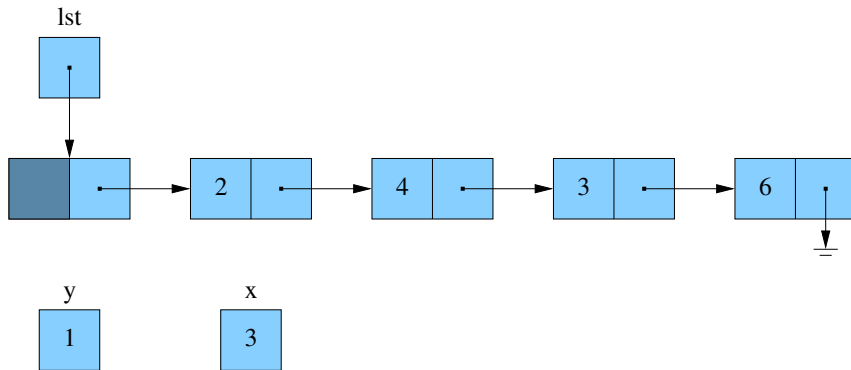
# Operações sobre listas lineares com cabeça

## ► busca seguida de inserção

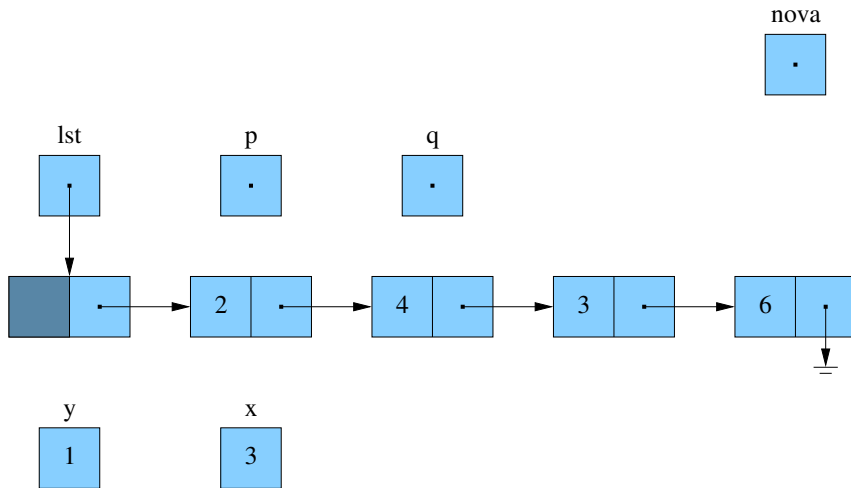
```
void busca_inserere_C(int y, int x, celula *lst)
{
    celula *p, *q, *nova;

    nova = (celula *) malloc(sizeof (celula));
    nova->chave = y;
    p = lst;
    q = lst->prox;
    while (q != NULL && q->chave != x) {
        p = q;
        q = q->prox;
    }
    nova->prox = q;
    p->prox = nova;
}
```

# Operações sobre listas lineares com cabeça

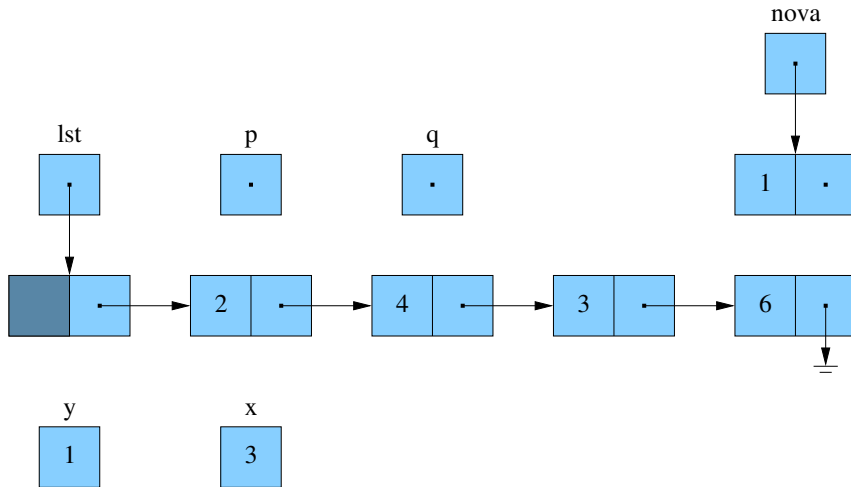


# Operações sobre listas lineares com cabeça

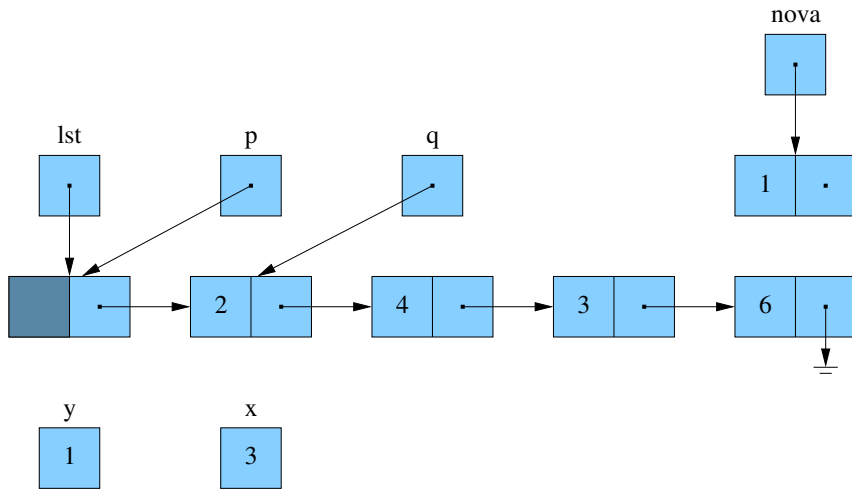




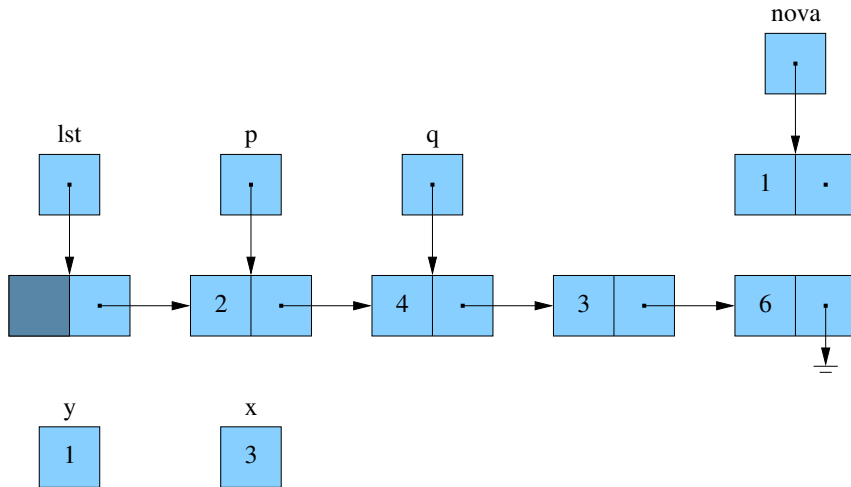
# Operações sobre listas lineares com cabeça



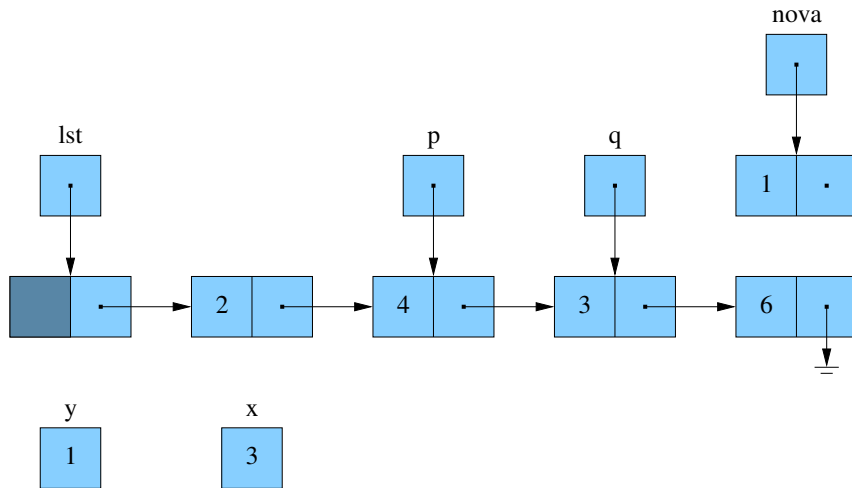
# Operações sobre listas lineares com cabeça



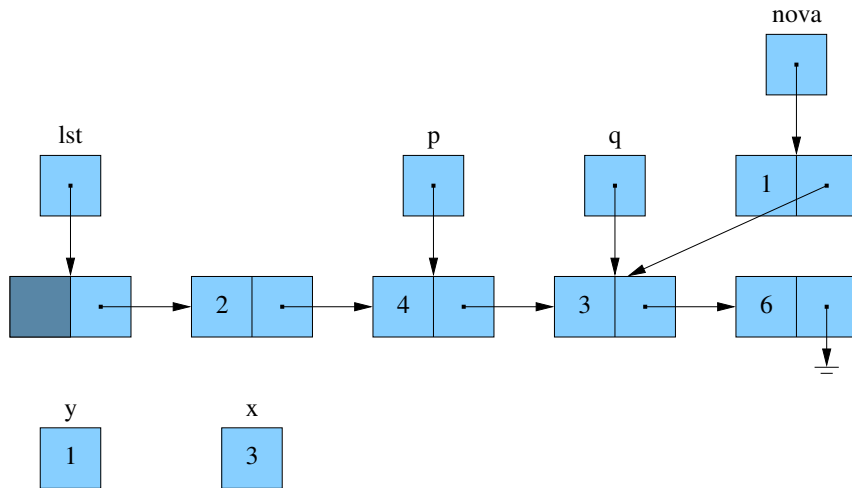
# Operações sobre listas lineares com cabeça



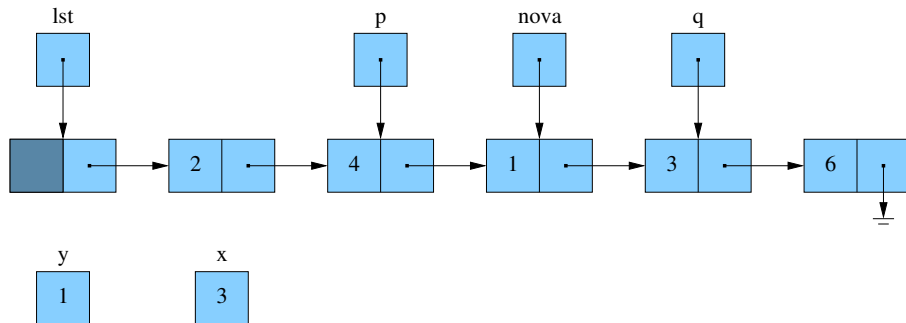
# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares com cabeça



# Operações sobre listas lineares sem cabeça

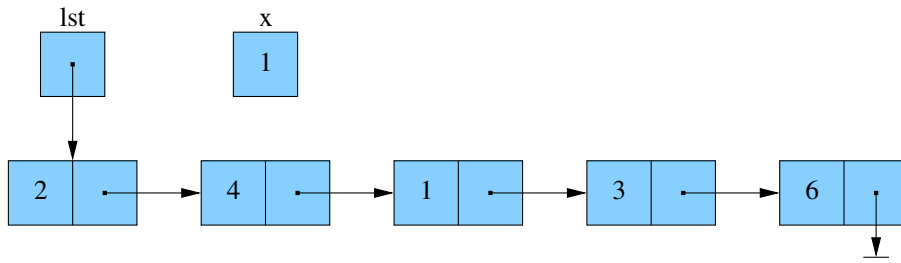
## ► busca não-recursiva

```
celula *busca_S(int x, celula *lst)
{
    celula *p;

    p = lst;
    while (p != NULL && p->chave != x)
        p = p->prox;

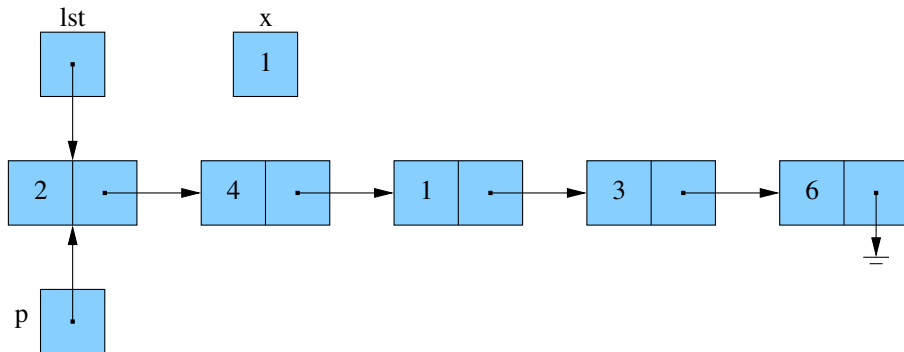
    return p;
}
```

# Operações sobre listas lineares sem cabeça

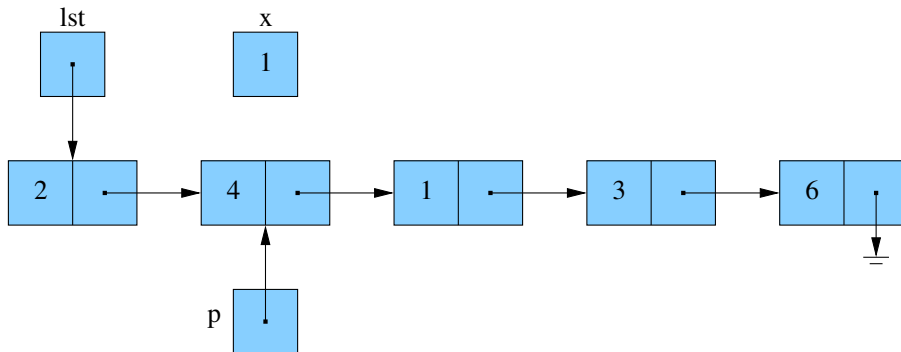




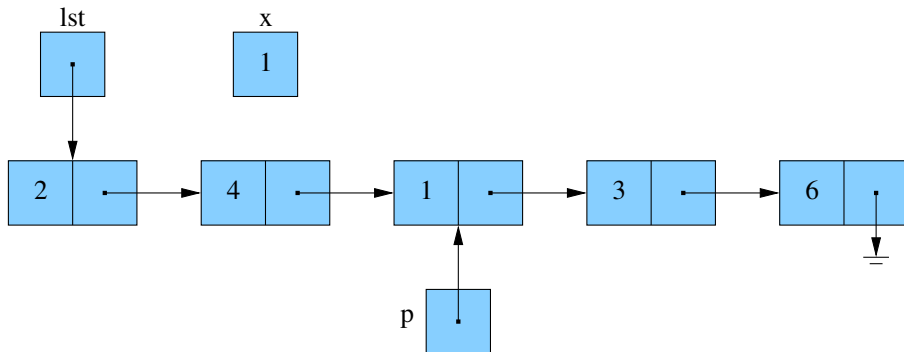
# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça

## ► busca recursiva

```
celula *buscaR_S(int x, celula *lst)
{
    if (lst == NULL)
        return NULL;

    if (lst->chave == x)
        return lst;

    return buscaR_S(x, lst->prox);
}
```

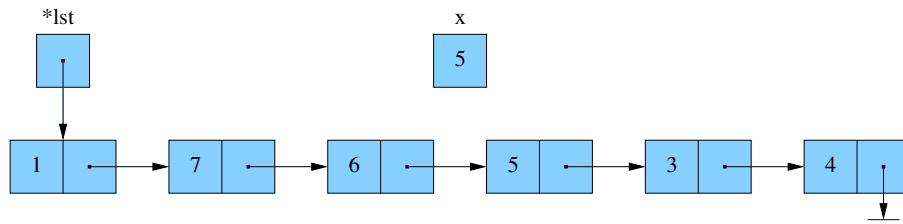
# Operações sobre listas lineares sem cabeça

## ► busca seguida de remoção

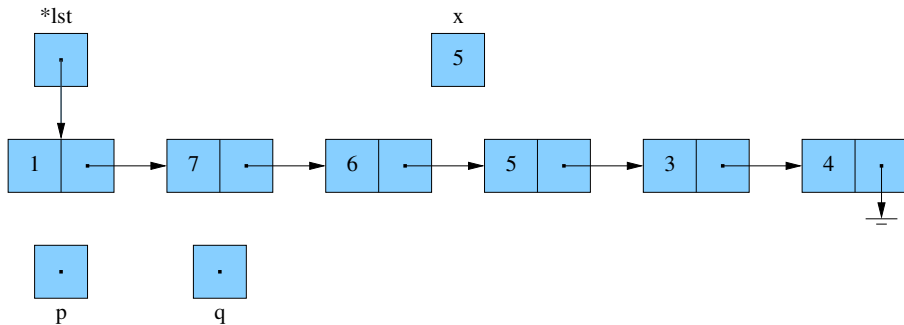
```
void busca_remove_S(int x, celula **lst)
{
    celula *p, *q;

    p = NULL;
    q = *lst;
    while (q != NULL && q->chave != x) {
        p = q;
        q = q->prox;
    }
    if (q != NULL) {
        if (p != NULL)
            p->prox = q->prox;
        else
            *lst = q->prox;
        free(q);
    }
}
```

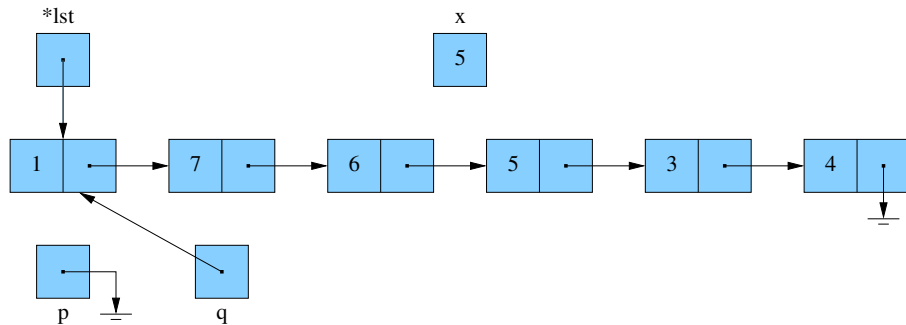
# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça

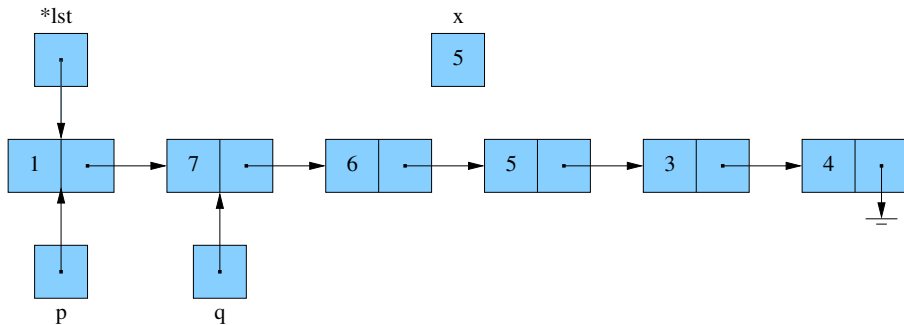


# Operações sobre listas lineares sem cabeça

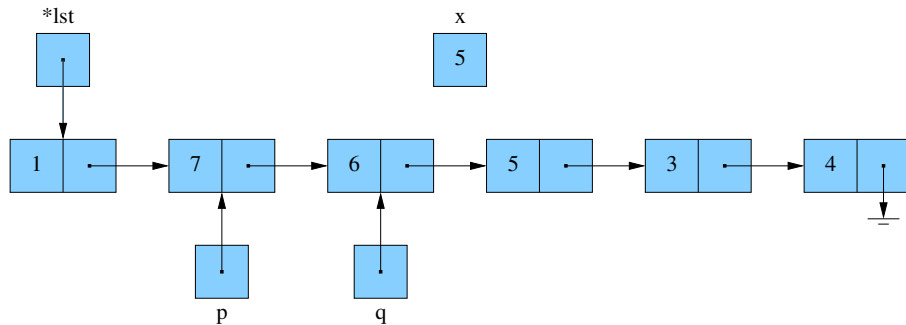




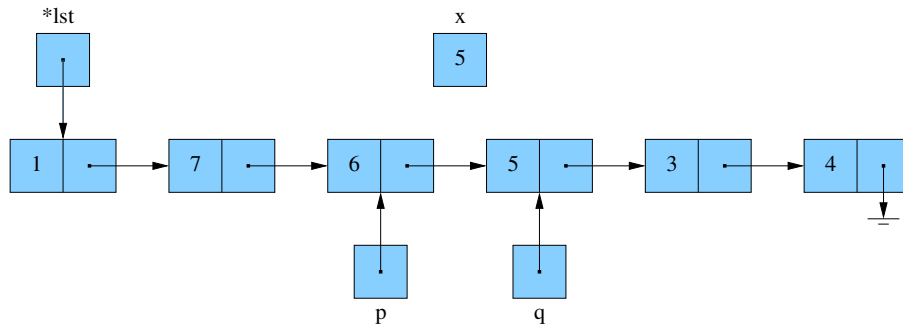
# Operações sobre listas lineares sem cabeça



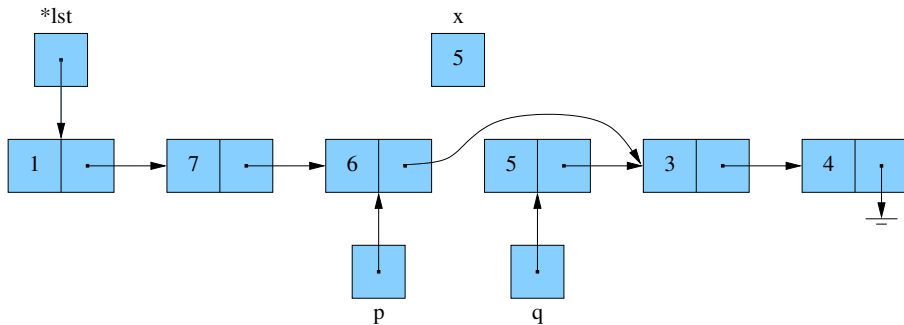
# Operações sobre listas lineares sem cabeça



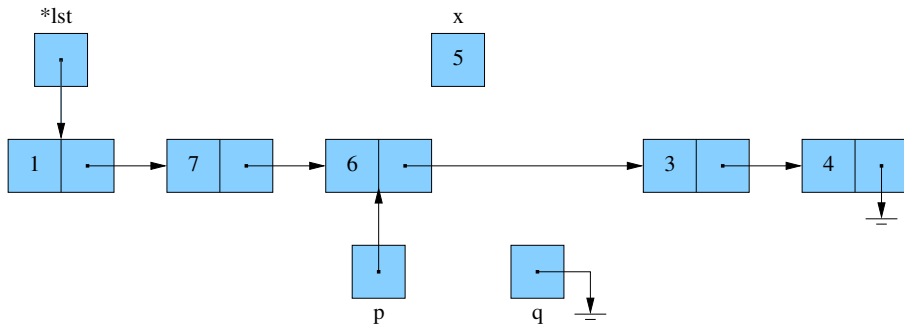
# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça

- ▶ uma chamada à função `busca_remove_S` é ilustrada abaixo, para um número inteiro `x` e uma `lista`:

```
busca_remove_S(x, &lista);
```

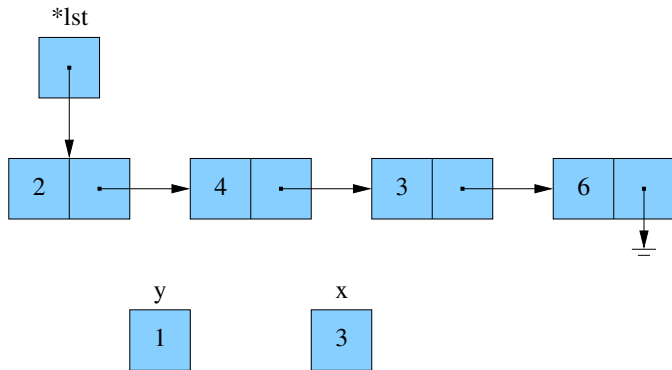
# Operações sobre listas lineares sem cabeça

## ► busca seguida de inserção

```
void busca_inserere_S(int y, int x, celula **lst)
{
    celula *p, *q, *nova;

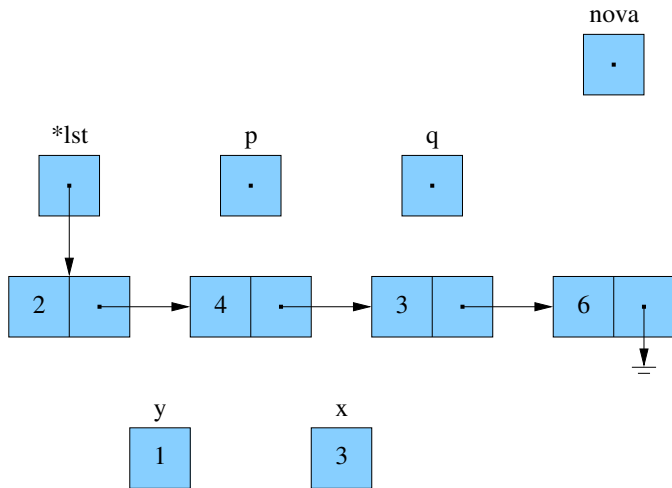
    nova = (celula *) malloc(sizeof (celula));
    nova->chave = y;
    p = NULL;
    q = *lst;
    while (q != NULL && q->chave != x) {
        p = q;
        q = q->prox;
    }
    nova->prox = q;
    if (p != NULL)
        p->prox = nova;
    else
        *lst = nova;
}
```

# Operações sobre listas lineares sem cabeça

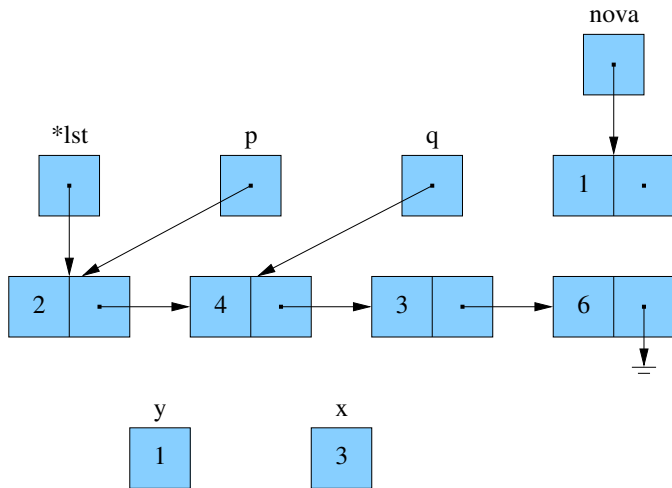




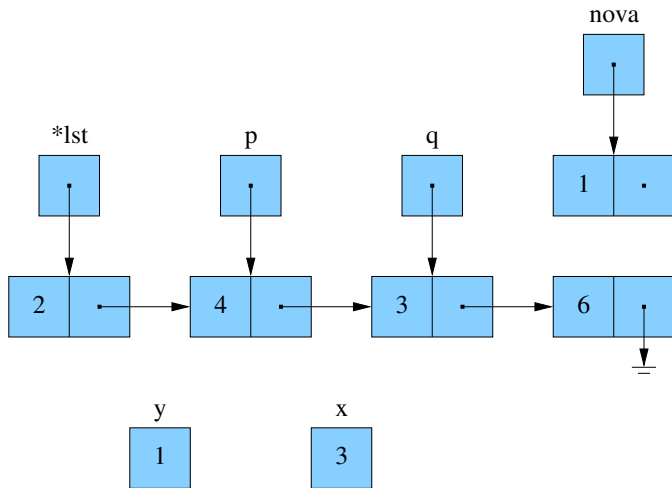
# Operações sobre listas lineares sem cabeça



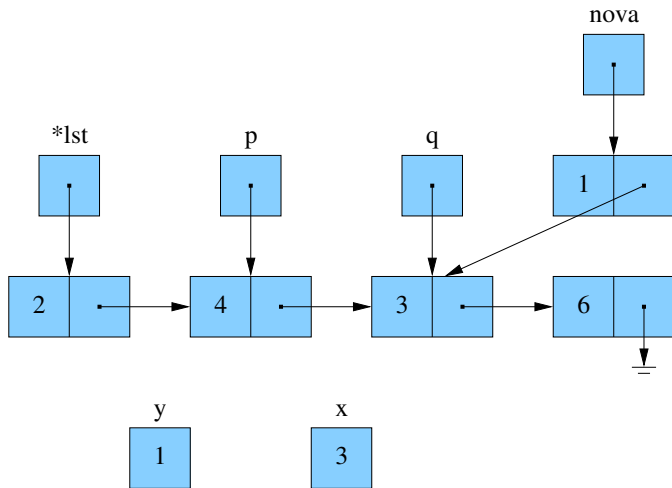
# Operações sobre listas lineares sem cabeça



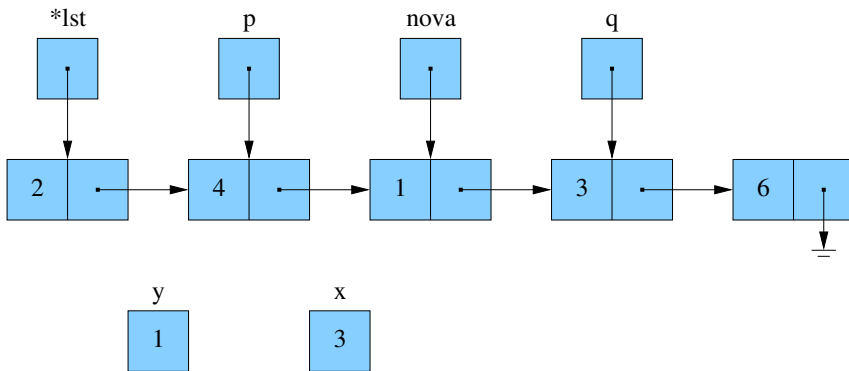
# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça



# Operações sobre listas lineares sem cabeça



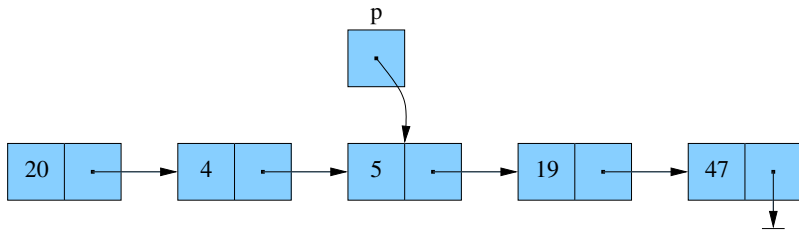
# Operações sobre listas lineares sem cabeça

- ▶ uma chamada à função `busca_insere_S` é ilustrada abaixo, para números inteiros `y` e `x` e uma `lista`:

```
busca_insere_S(y, x, &lista);
```

# Exercícios

1. Se conhecemos apenas o ponteiro **p** para uma célula de uma lista linear em alocação encadeada, como na figura abaixo, e nada mais é conhecido, como podemos modificar a lista linear de modo que passe a conter apenas os valores 20, 4, 19, 47, isto é, sem o conteúdo da célula apontada por **p**?



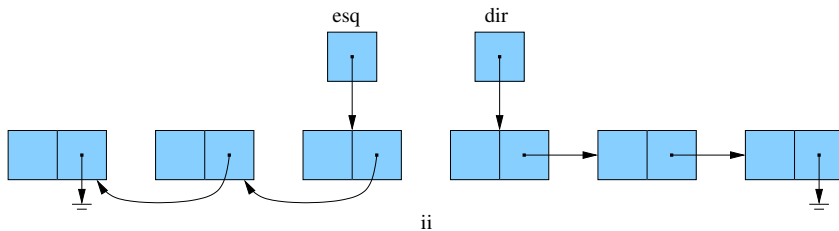
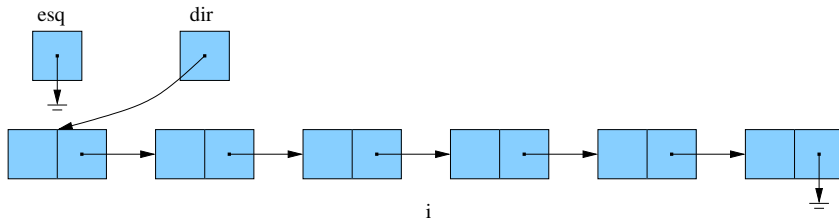
2. O esquema apresentado na figura abaixo permite percorrer uma lista linear encadeada nos dois sentidos, usando apenas o campo **prox** que contém o endereço do próximo elemento da lista. Usamos dois ponteiros **esq** e **dir**, que apontam para dois elementos vizinhos da lista. A idéia desse esquema é que à medida que os ponteiros **esq** e **dir** caminham na lista, os campos **prox** são invertidos de maneira a permitir o tráfego nos dois sentidos.

Escreva funções para:

- (a) mover **esq** e **dir** para a direita de uma posição
- (b) mover **esq** e **dir** para a esquerda de uma posição



# Exercícios



3. Que acontece se trocarmos `while (p != NULL && p->chave != x)` por `while (p->chave != x && p != NULL)` na função `busca_C`?
4. Escreva uma função que encontre uma célula cuja chave tem valor mínimo em uma lista linear encadeada. Considere listas com e sem cabeça e escreva versões não-recursivas e recursivas para a função
5. Escreva uma função `busca_inserere_fim` que receba um número inteiro `y`, uma lista linear encadeada `lst` e um ponteiro `f` para o fim da lista e realize a inserção desse valor no final da lista. Faça uma versões para listas lineares com cabeça e sem cabeça.

6. (a) Escreva duas funções: uma que copie um vetor para uma lista linear encadeada com cabeça; outra, que faça o mesmo para uma lista linear sem cabeça.  
(b) Escreva duas funções: uma que copie uma lista linear encadeada com cabeça em um vetor; outra que copie uma lista linear encadeada sem cabeça em um vetor.
7. Escreva uma função que decida se duas listas dadas têm o mesmo conteúdo. Escreva duas versões: uma para listas lineares com cabeça e outra para listas lineares sem cabeça.
8. Escreva uma função que conte o número de células de uma lista linear encadeada.

9. Seja **lista** uma lista linear com seus conteúdos dispostos em ordem crescente. Escreva funções para realização das operações básicas de busca, inserção e remoção, respectivamente, em uma lista linear com essa característica. Escreva conjuntos de funções distintas para listas lineares com cabeça e sem cabeça. As operações de inserção e remoção devem manter a lista em ordem crescente.
10. Sejam duas listas lineares **lst1** e **lst2**, com seus conteúdos dispostos em ordem crescente. Escreva uma função **concatena** que receba **lst1** e **lst2** e construa uma lista **R** resultante da intercalação dessas duas listas, de tal forma que a lista construída também esteja ordenada. A função **concatena** deve destruir as listas **lst1** e **lst2** e deve devolver **R**. Escreva duas funções para os casos em que as listas lineares encadeadas são com cabeça e sem cabeça.

11. Seja **lst** uma lista linear encadeada composta por células contendo os valores  $c_1, c_2, \dots, c_n$ , nessa ordem. Para cada item abaixo, escreva duas funções considerando que **lst** é com cabeça e sem cabeça.
- (a) Escreva uma função **roda1** que receba uma lista e modifique e devolva essa lista de tal forma que a lista resultante contenha as chaves  $c_2, c_3, \dots, c_n, c_1$ , nessa ordem.
  - (b) Escreva uma função **inverte** que receba uma lista e modifique e devolva essa lista de tal forma que a lista resultante contenha as chaves  $c_n, c_{n-1}, \dots, c_2, c_1$ , nessa ordem.
  - (c) Escreva uma função **soma** que receba uma lista e modifique e devolva essa lista de tal forma que a lista resultante contenha as chaves  $c_1 + c_n, c_2 + c_{n-1}, \dots, c_{n/2} + c_{n/2+1}$ , nessa ordem. Considere  $n$  par.

12. Sejam  $S_1$  e  $S_2$  dois conjuntos disjuntos de números inteiros. Suponha que  $S_1$  e  $S_2$  estão implementados em duas listas lineares em alocação encadeada. Escreva uma função **uniao** que receba as listas representando os conjuntos  $S_1$  e  $S_2$  e devolva uma lista resultante que representa a união dos conjuntos, isto é, uma lista linear encadeada que representa o conjunto  $S = S_1 \cup S_2$ . Considere os casos em que as listas lineares encadeadas são com cabeça e sem cabeça.

13. Seja um polinômio  $p(x) = a_0x^n + a_1x^{n-1} + \dots + a_n$ , com coeficientes de ponto flutuante. Represente  $p(x)$  adequadamente por uma lista linear encadeada e escreva as seguintes funções. Para cada item a seguir, considere o caso em que a lista linear encadeada é com cabeça e sem cabeça.
- (a) Escreva uma função **pponto** que receba uma lista **p** e um número de ponto flutuante **x0** e calcule e devolva  $p(x_0)$ .
  - (b) Escreva uma função **psoma** que receba as listas lineares **p** e **q**, que representam dois polinômios, e calcule e devolva o polinômio resultante da operação  $p(x) + q(x)$ .
  - (c) Escreva uma função **pprod** que receba as listas lineares **p** e **q**, que representam dois polinômios, e calcule e devolva o polinômio resultante da operação  $p(x) \cdot q(x)$ .

14. Escreva uma função que aplique a função **free** a todas as células de uma lista linear encadeada, supondo que todas as suas células foram alocadas com a função **malloc**. Faça versões considerando listas lineares encadeadas com cabeça e sem cabeça.



