Listas Lineares

Algoritmos e Programação II

(slides baseados na apostila do Prof Fábio Viduani)

Conteúdo da aula

- Introdução
- Definição
- Tipos de listas
- Operações sobre listas lineares sem cabeça

- Os slides sobre este assunto de listas lineares são parcialmente baseados no segundo capítulo do livro D. E. Knuth. The Art of Computer Programming. Volume 1, Addison Wesley, 1973.

Introdução

- Uma estrutura de dados armazena dados na memória do computador a fim de permitir o **acesso eficiente** dos mesmos.
- A maioria das estruturas de dados consideram a memória primária (a chamada RAM) para armazenamento, tais como pilhas, filas, árvores binárias de busca, árvores AVL e árvores rubro-negras. Outras são especialmente projetadas e adequadas para serem armazenadas em memórias secundárias como o disco rígido, i.e., B-árvores.
- Uma estrutura de dados bem projetada permite a manipulação eficiente, em tempo e em espaço, dos dados armazenados através de operações específicas.

Introdução

- Uma estrutura de dados armazena dados na memória do computador a fim de permitir o **acesso eficiente** dos mesmos.
- Pilhas, filas, árvores binárias de busca, árvores AVL e árvores rubro-negras, B-árvores.
- Um conceito relacionado com a estrutura de dados é o **Tipo Abstrato de Dados (TAD)**.
- **Um Tipo Abstrato é o conjunto de dados e de operações sobre esses dados.**
- Exemplo sem maiores detalhes.
  - Tipo Stack (PILHA)
  - Operações:
    - 1 Push(Empilhar);
    - 2 Pop(Desempilhar);
    - 3 Top(Topo);
    - 4 Clear;
    - 5 Empty

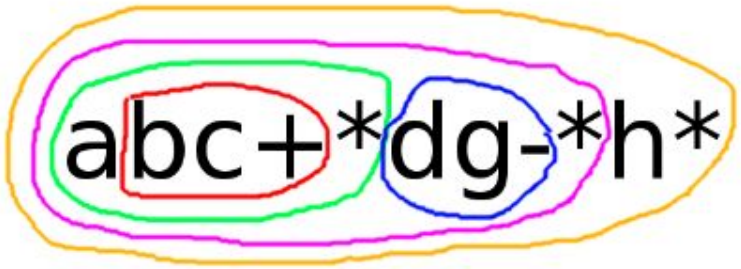
Introdução

- Lista linear: a primeira estrutura de dados que aprendemos
- Diversas aplicações importantes para organização de informações na memória tais como:
  - representações alternativas para expressões aritméticas

Exemplos:

infixa	pós-fixa
a-b	ab-
a-b*c	abc*-
(a-b)*c	ab-c*
a+b*c^d-e	abcd^*+e-
a*(b+c)*(d-g)*h	abc+*dg-*h*
a*b-c*d^e/f+g*h	ab*cde^*f/-gh*+

infixa	posfixa
5 * (((9 + 8) * (4 * 6)) + 7)	5 9 8 + 4 6 * * 7 + *



Introdução

- Lista linear: a primeira estrutura de dados que aprendemos

- Diversas aplicações importantes para organização de informações na memória tais como:
  - representações alternativas para expressões aritméticas
  - armazenamento de argumentos de funções

    void identificador_funcao(param1, param2, param3,…)

  - 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;
- **Exemplos**: 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, chamado de **célula ou NÓ**;

Definição

- 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 um conjunto de $n > 0$ **células** (***nós***) $c_1, c_2, \ldots, c_n$ 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, inserção e remoção.

- dependendo da aplicação, muitas outras operações também podem ser realizadas sobre essa estrutura

Definição

- Algumas operações que podemos querer realizar sobre listas lineares:
  - Ter acesso a c_k , k qualquer, a fim de examinar ou alterar o conteúdo de seus campos
  - Inserir um elemento novo antes ou depois de c_k
  - Remover c_k
  - Colocar todos os elementos da lista em ordem.
  - Combinar 2 ou mais listas lineares em uma só
  - Quebrar uma lista linear em duas ou mais
  - Copiar uma lista linear em um outro espaço
  - Trataremos as três primeiras operações, para k=1 e k=n, as listas recebem nomes (pilha ou fila) conforme a maneira que tais operações são realizadas.

Definição

- 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;

Definição

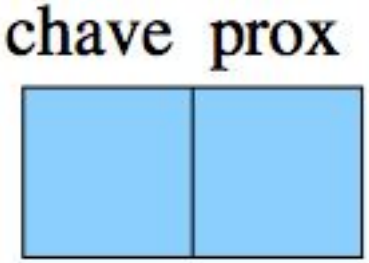
- o problema que queremos resolver é que define o tipo de armazenamento a ser usado, dependendo:
  - das operações sobre a lista,
  - do número de listas envolvidas e
  - das características particulares das listas.

- já vimos as operações básicas sobre uma lista linear em alocação sequencial (VETOR);

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 (**nó**) da lista indicando o endereço do próximo elemento da lista



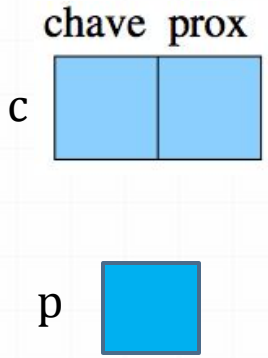
Definição em C++

- Definição do tipo célula/nó de uma lista linear encadeada:

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

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

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



Definição

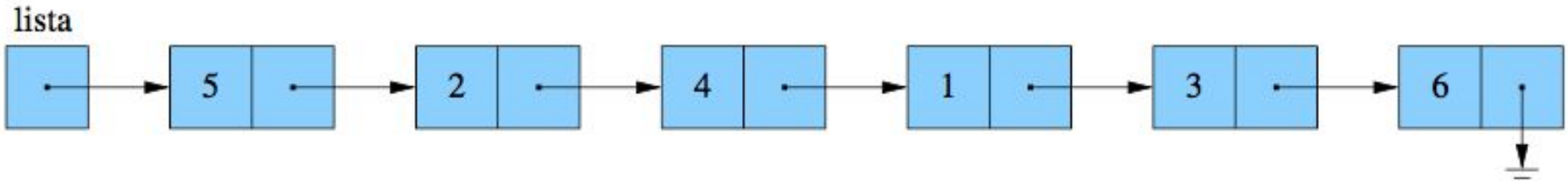
- se **c** é uma célula (struct *celula*) então **c.chave** é o conteúdo da célula e **c.prox** é o endereço da célula seguinte

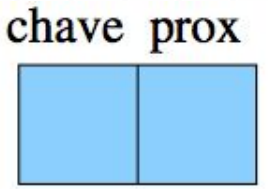
- se **p** é o endereço de uma célula (*celula*) então **p->chave** é o valor do atributo chave da célula apontada por **p** e **p->prox** é o endereço da célula seguinte

p = &c;

- seja **lista** o ponteiro para o primeiro nó de uma lista encadeada.
- se **p** é o endereço da última célula (*celula*) da lista então **p->prox** vale **NULL**

Exemplo





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"

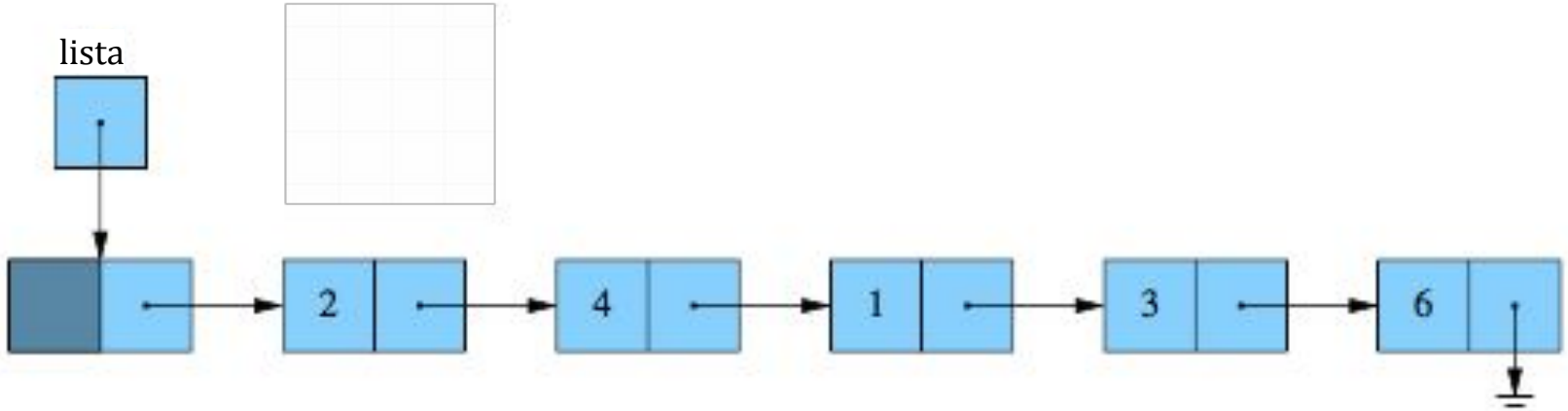
Definição

- uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa;

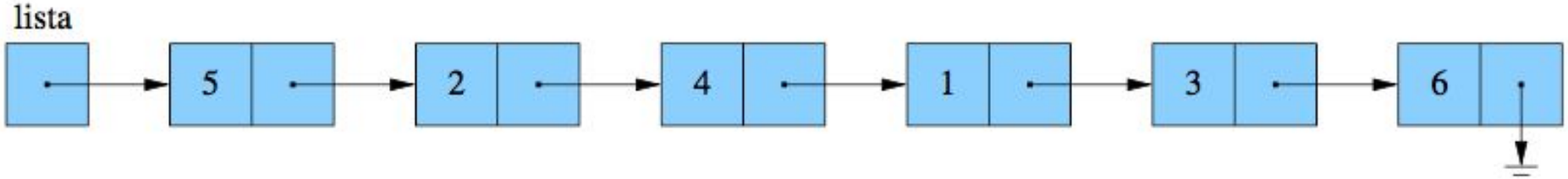
- em uma lista linear **com cabeça**, a primeira célula serve apenas para marcar o início da lista e portanto, o seu conteúdo é irrelevante; a primeira célula é a **cabeça** da lista;

- em uma lista linear **sem cabeça** o conteúdo da primeira célula é tão relevante quanto o das demais;

LISTA LINEAR DINÂMICA COM CABEÇA



LISTA LINEAR DINÂMICA SEM CABEÇA



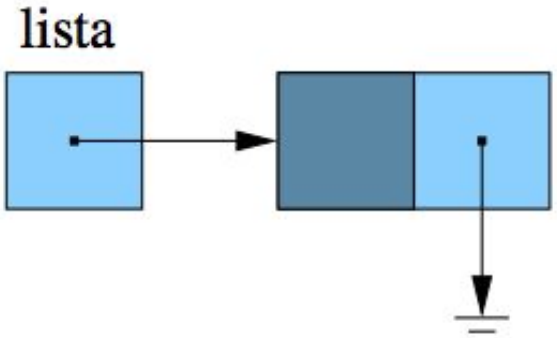
Definição

- uma lista linear está vazia se não tem célula alguma
- para criar uma lista vazia **com cabeça** "**lista**", 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;
```

- Trabalharemos com listas encadeadas SEM CABEÇA.

Operações sobre listas lineares sem cabeça

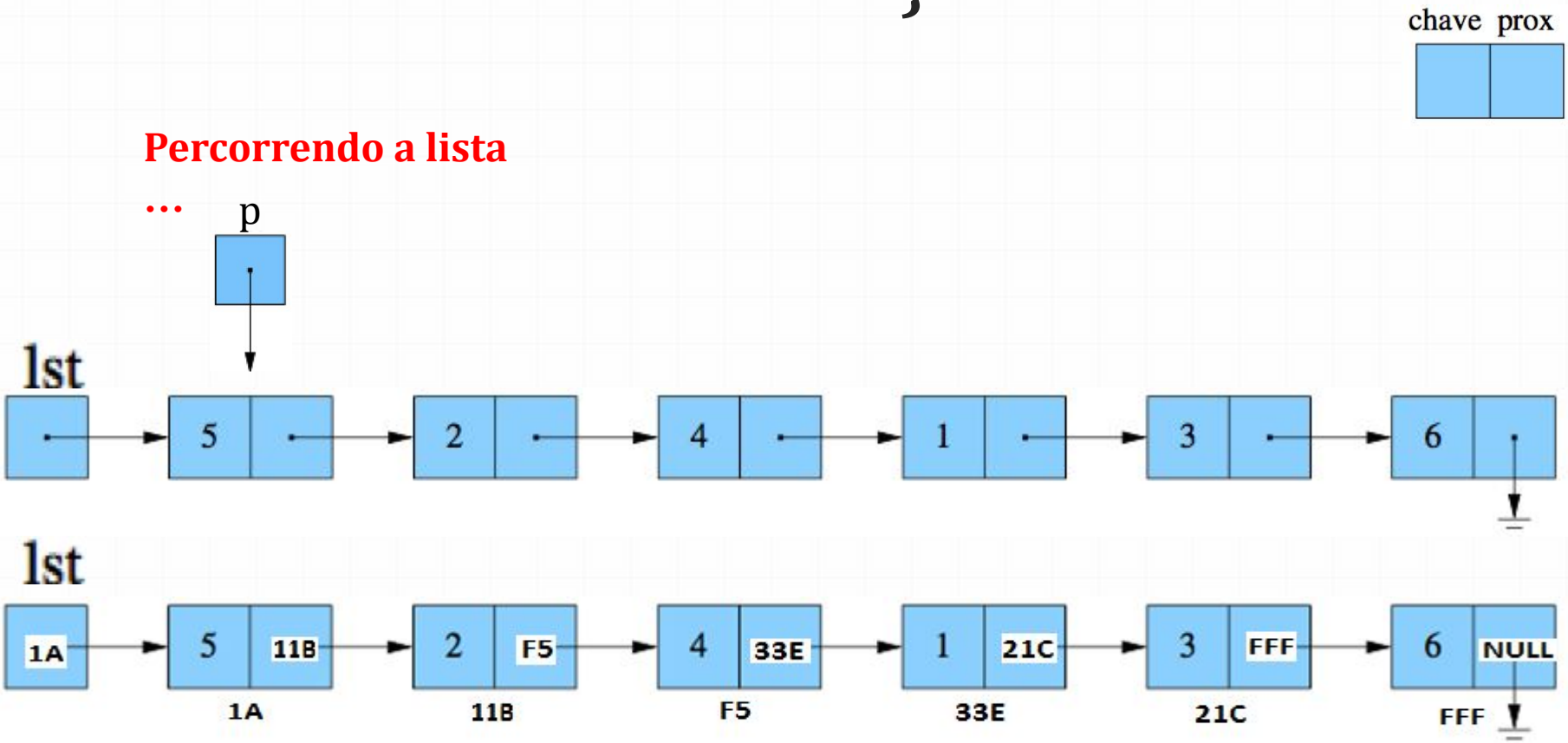
- para imprimir o conteúdo de todas as células (nós) de uma lista linear podemos usar a seguinte função:

```
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 sem cabeça, a chamada da função deve ser:

imprime_lista(lista);

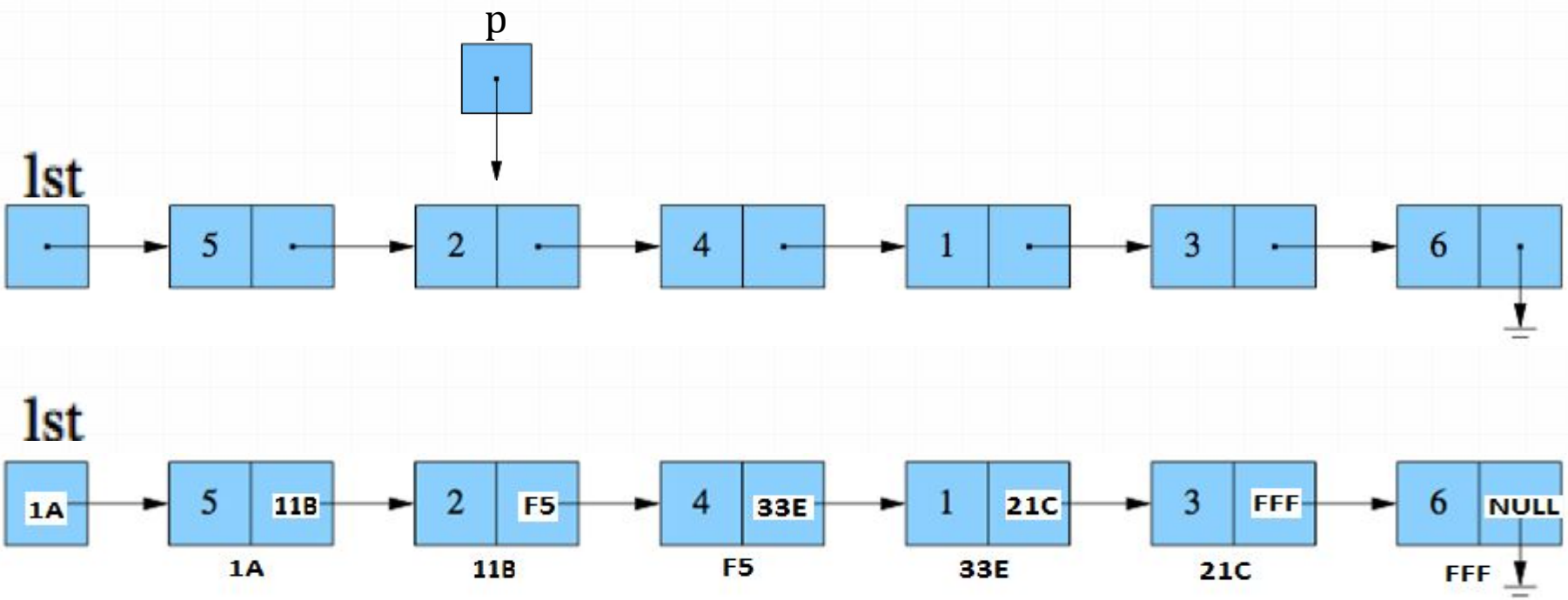
Operações sobre listas lineares sem cabeça



Operações sobre listas lineares sem cabeça

Percorrendo a lista

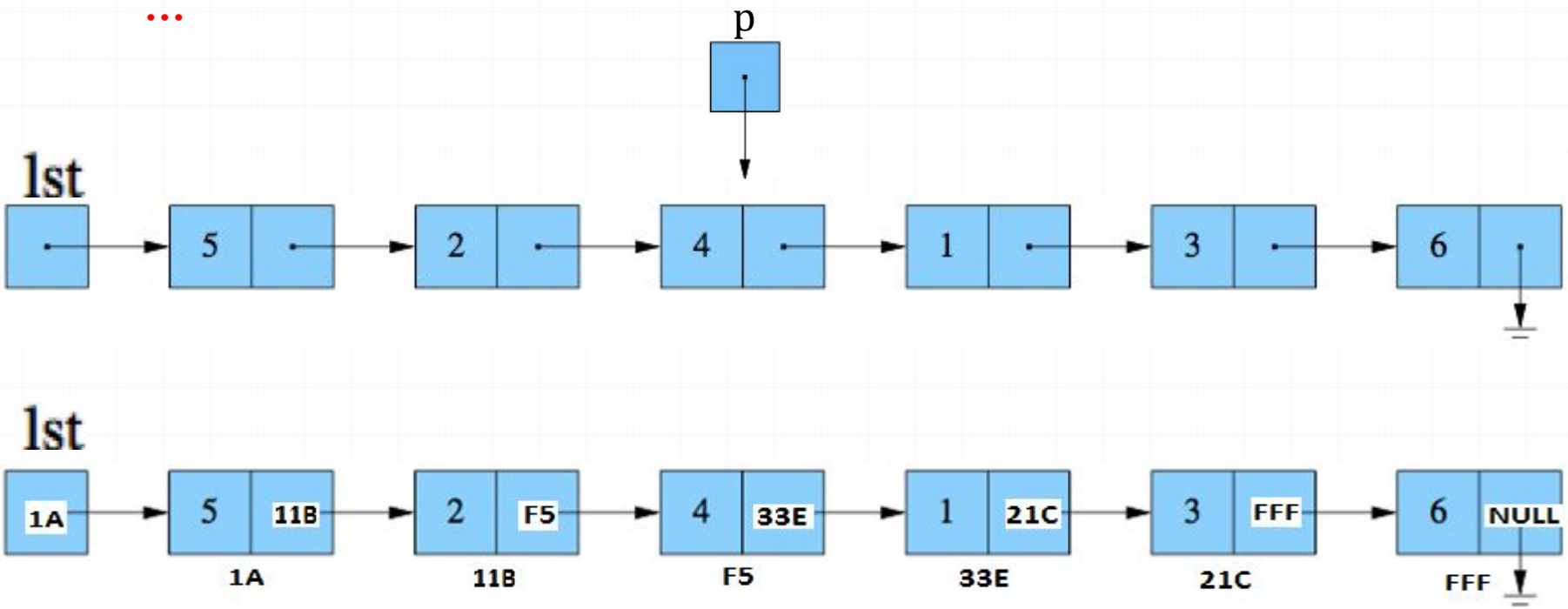
…



Operações sobre listas lineares sem cabeça

Percorrendo a lista

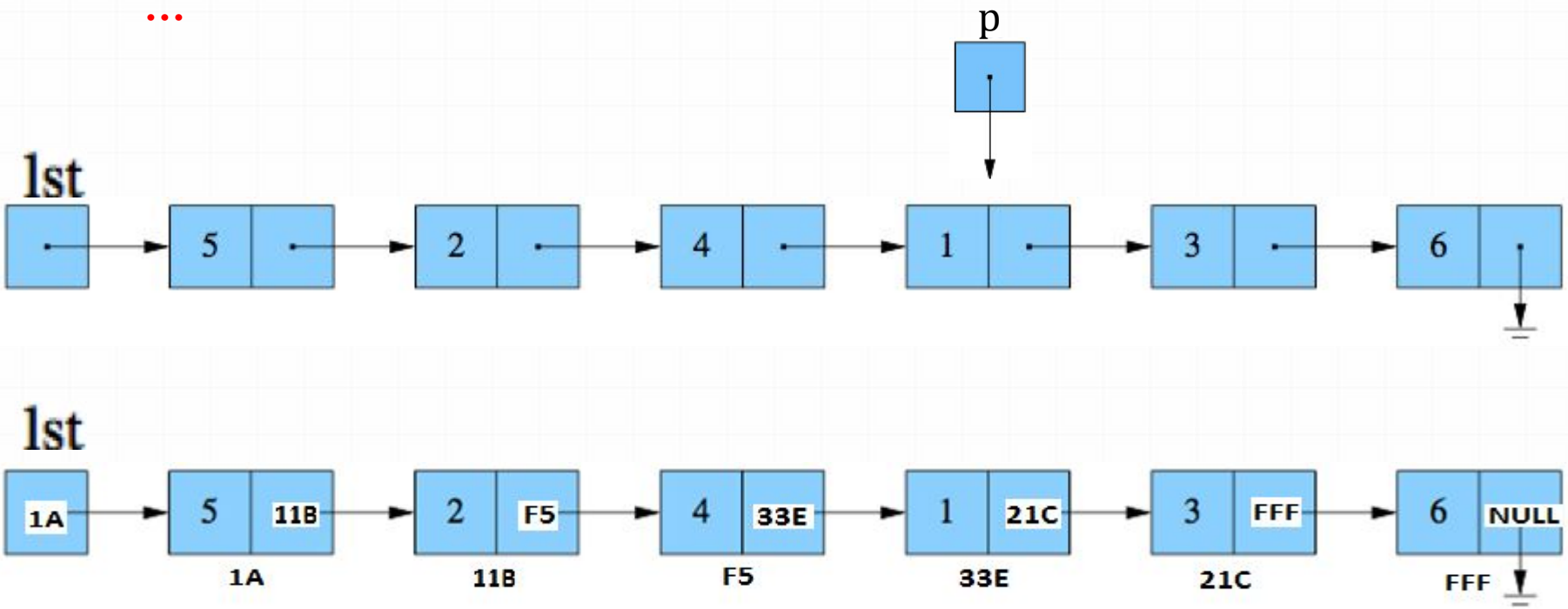
…



Operações sobre listas lineares sem cabeça

Percorrendo a lista

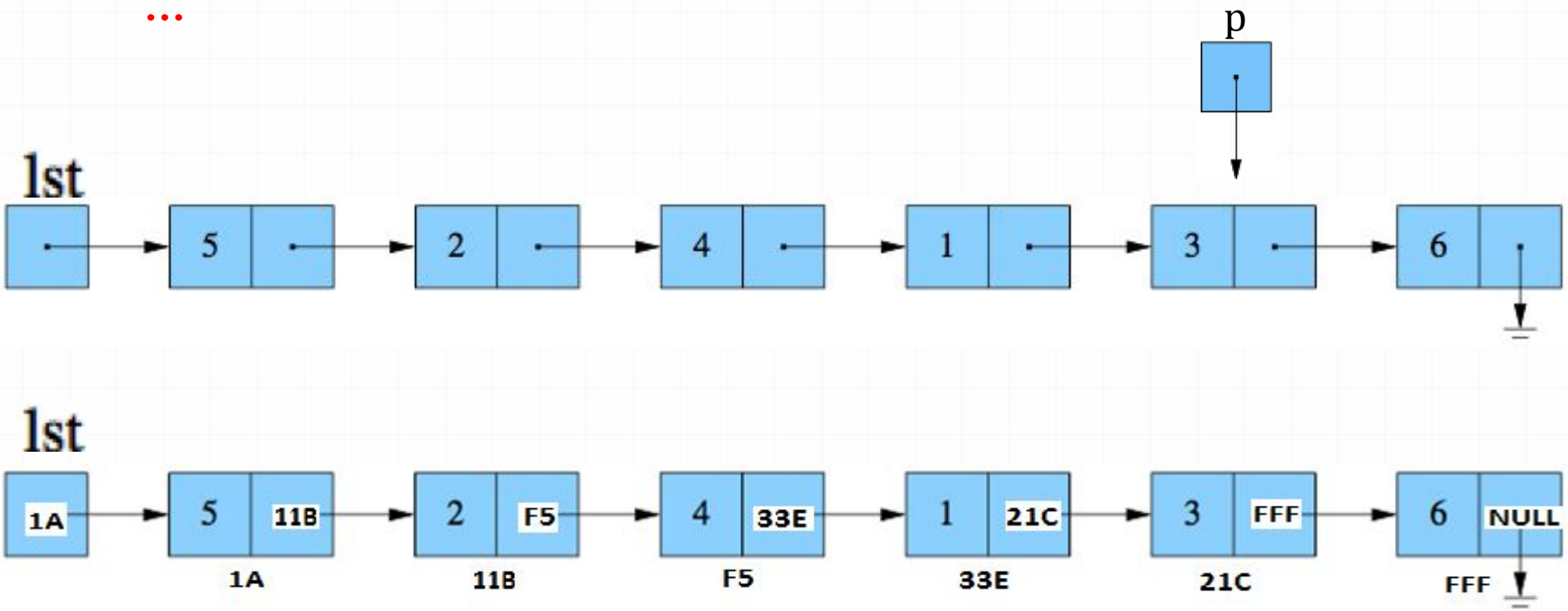
…



Operações sobre listas lineares sem cabeça

Percorrendo a lista

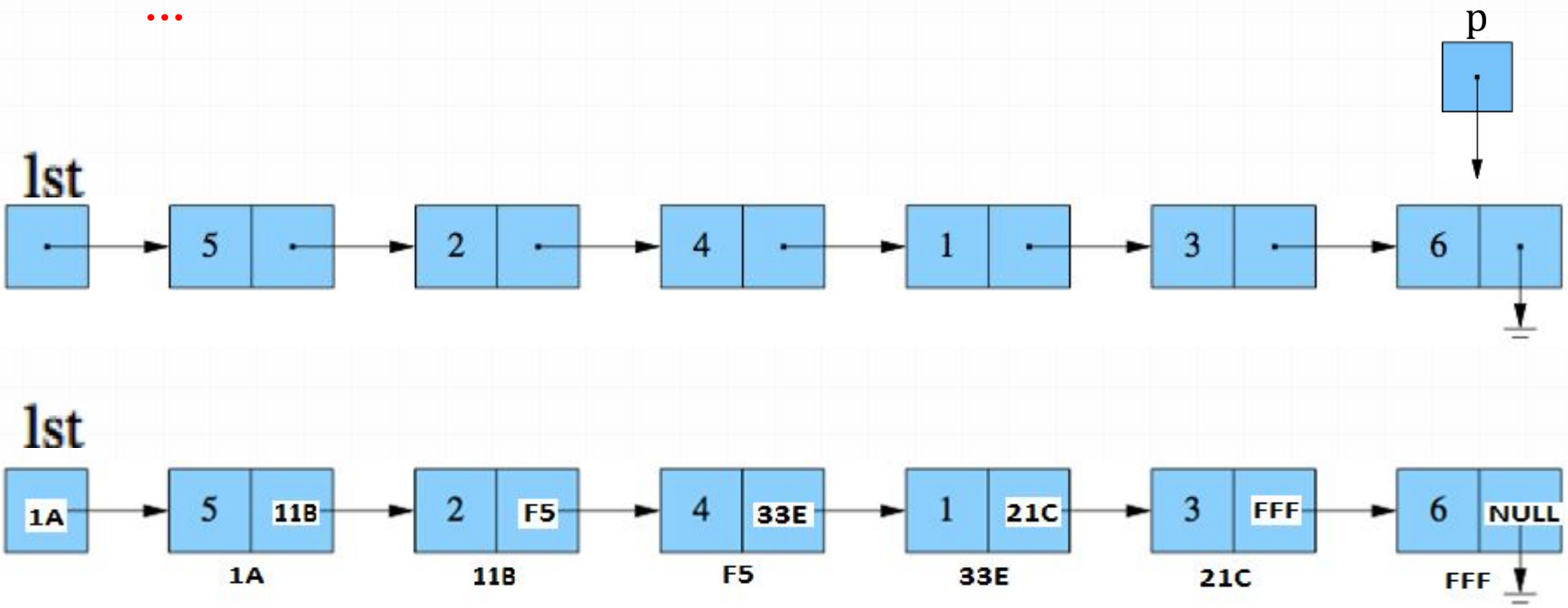
…



Operações sobre listas lineares sem cabeça

Percorrendo a lista

…

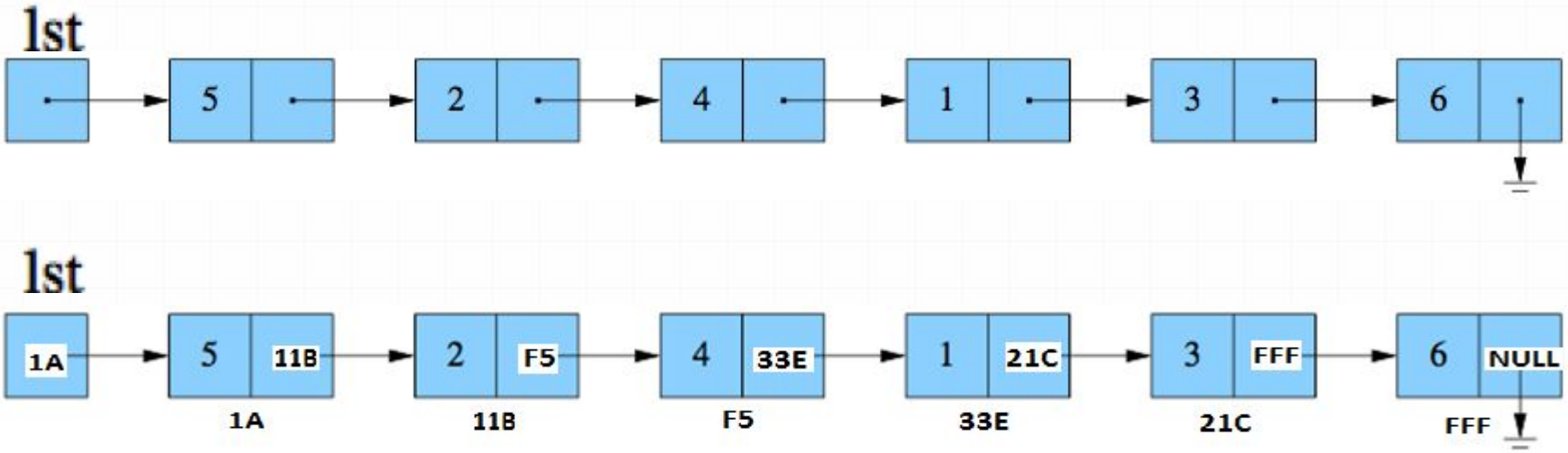


Operações sobre listas lineares sem cabeça

Percorrendo a lista

…

p = NULL



Operações sobre listas lineares sem cabeça

- para imprimir o conteúdo de todas as células (nós) 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 sem cabeça, a chamada da função deve ser:

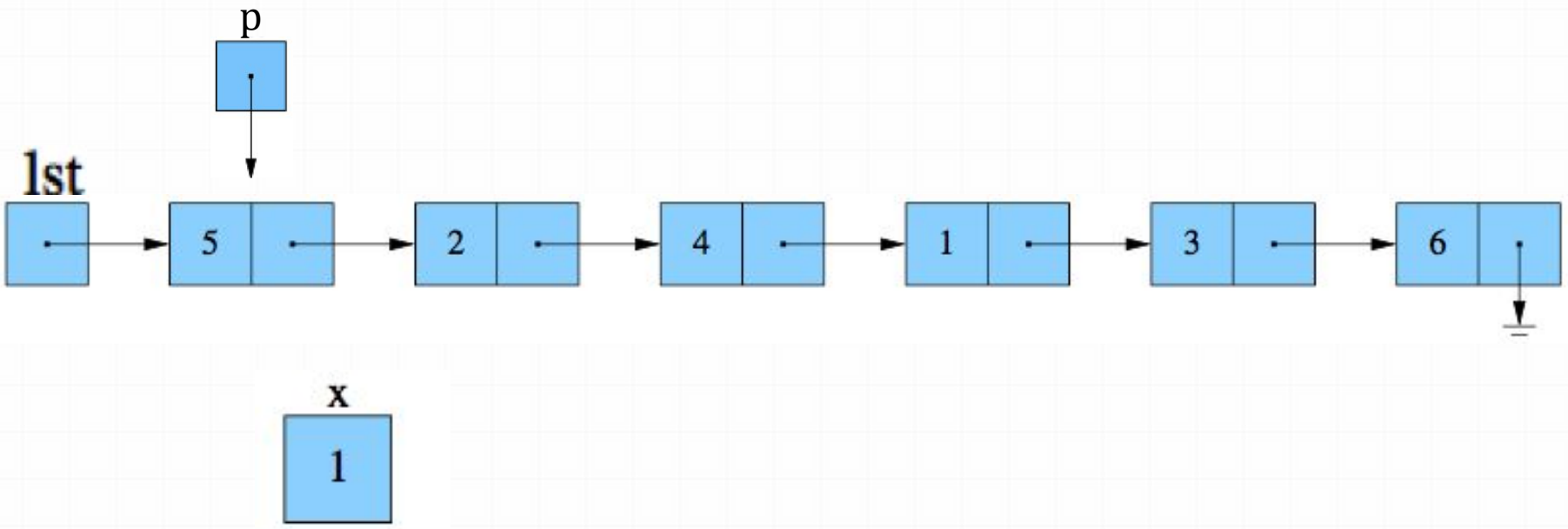
imprime_lista(lista);

Operações sobre listas lineares sem cabeça

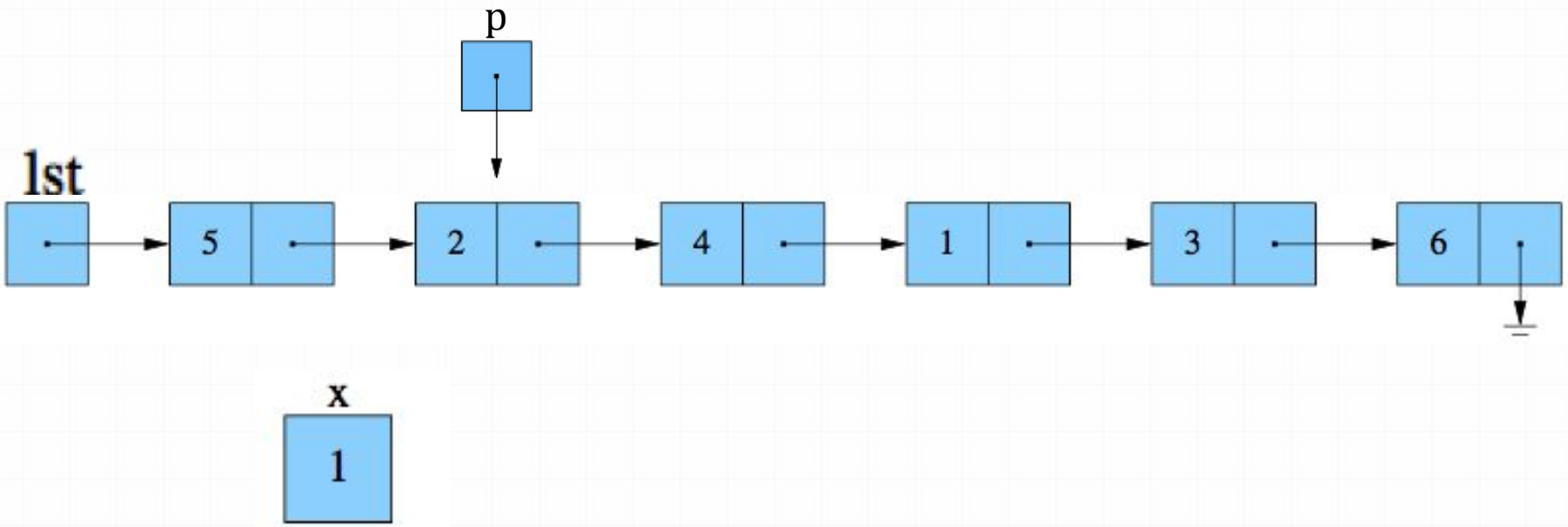
- Busca não-recursiva

```
/* Recebe um número inteiro x e uma lista encadeada sem cabeça
lst e devolve o endereço da célula que contém x ou NULL se tal
célula não existe */
celula* busca_S(int x, celula *lst)
{
    celula *p;
    p = lst;
    while (p != NULL && p->chave != x)
        p = p->prox;
    return p;
}
```

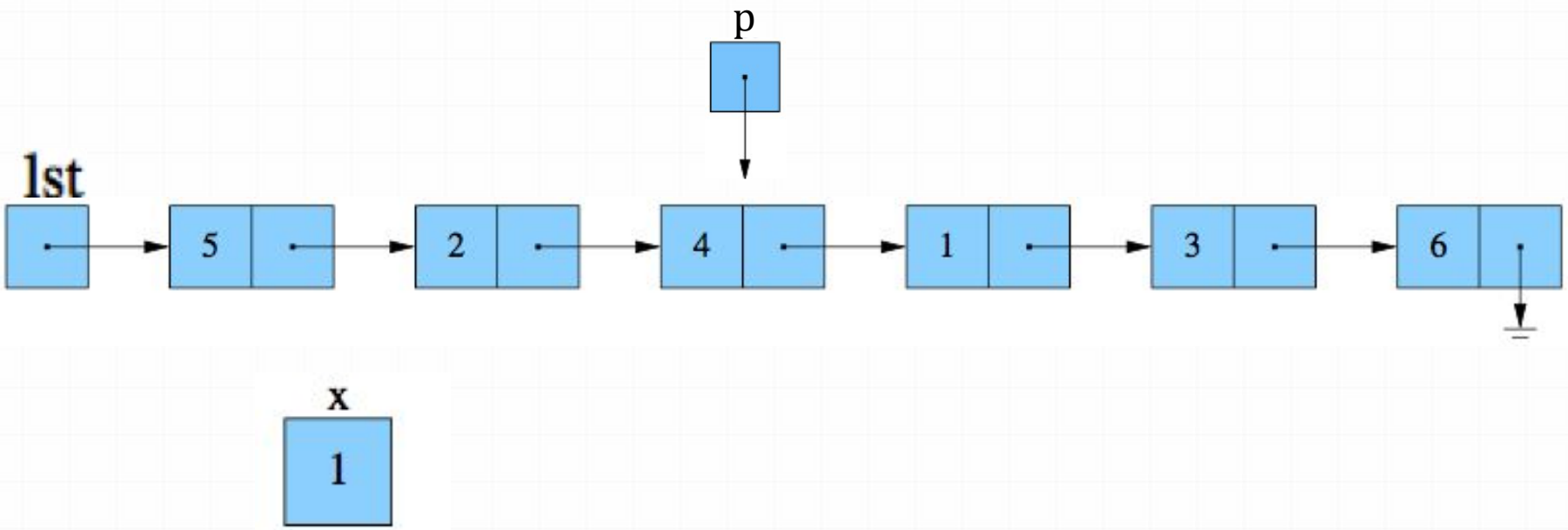
Operações sobre listas lineares sem cabeça



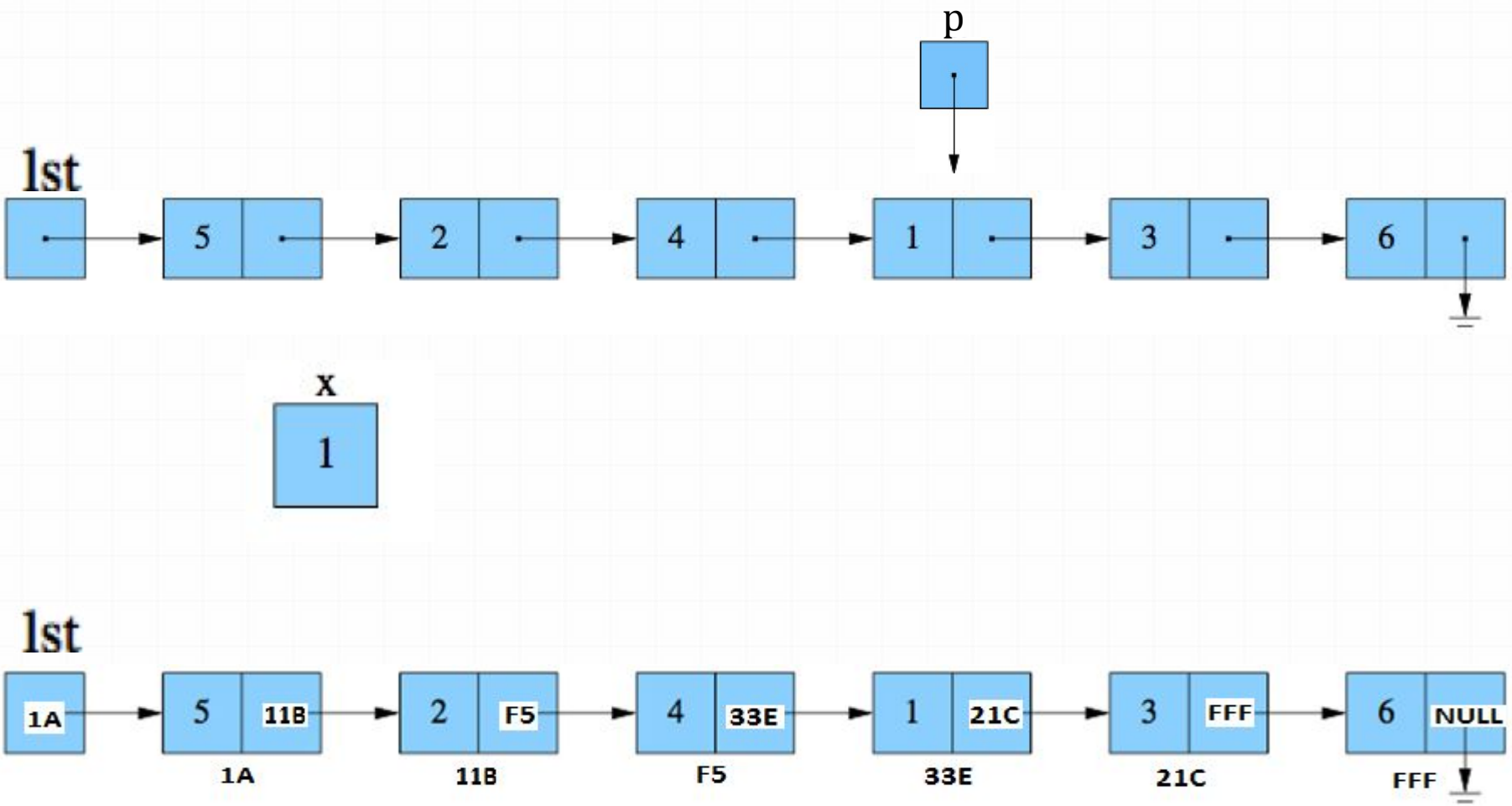
Operações sobre listas lineares sem cabeça



Operações sobre listas lineares sem cabeça



Operações sobre listas lineares sem cabeça



Operações sobre listas lineares sem cabeça

- Busca não-recursiva

```
/* Recebe um número inteiro x e uma lista encadeada sem cabeça
lst e devolve o endereço da célula que contém x ou NULL se tal
célula não existe */
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

- Busca recursiva

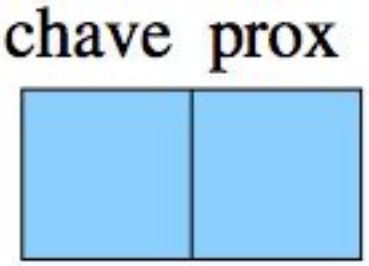
```
/* Recebe um número inteiro x e uma lista encadeada
sem cabeça lst e devolve o endereço da célula que
contém x ou NULL se tal célula não existe */
celula* buscaR_S(int x, celula *lst)
{
    if (lst == NULL)
        return NULL;
    if (lst->chave == x)
        return lst;
    return buscaR_S(x, lst->prox);
}
```

Operações sobre listas lineares sem cabeça

- Inserção de um elemento **x** na lista:
  - Pode ser de várias formas:
    - **Inserção no fim**
    - **Inserção no início**
    - **Inserção ordenada (entre outras)**

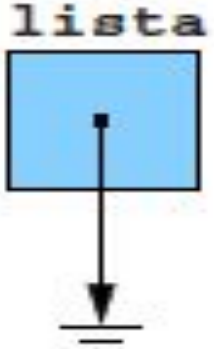
Inserção de x no fim da lista

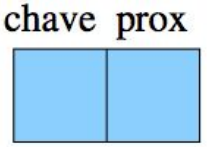
```
struct celula {
      int chave;
      struct celula *prox;
};
```

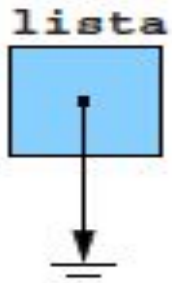


0 **Criação de uma lista encadeada vazia**

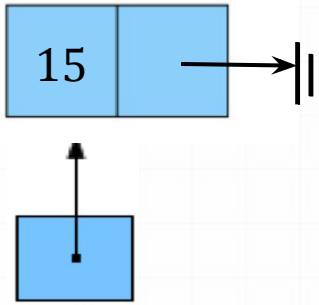
```
celula *lista = NULL;
```



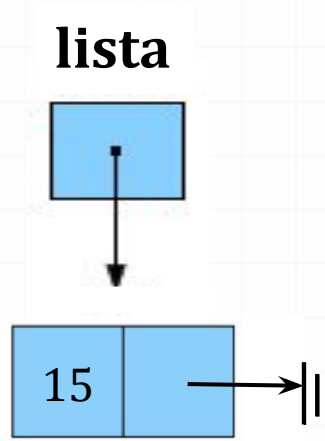


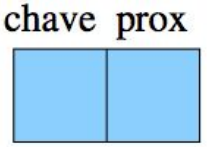


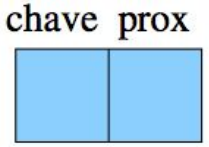
X = 15

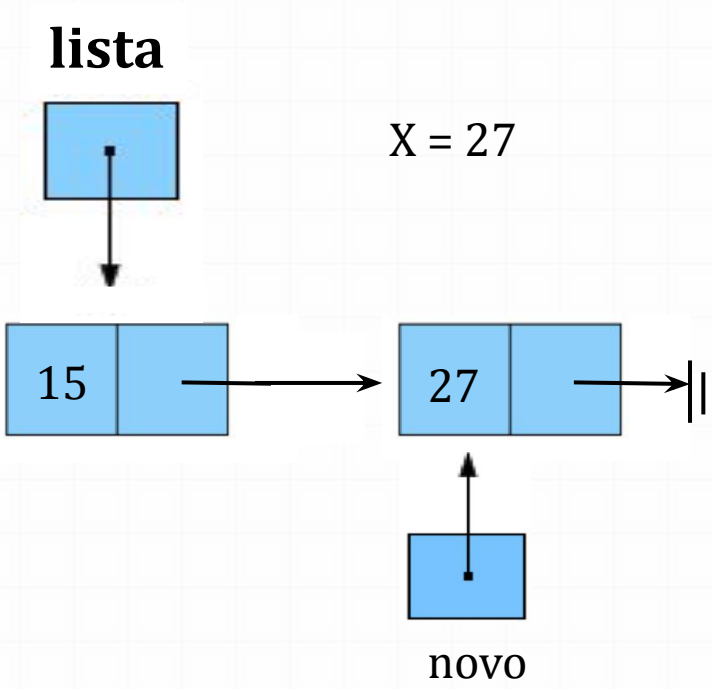


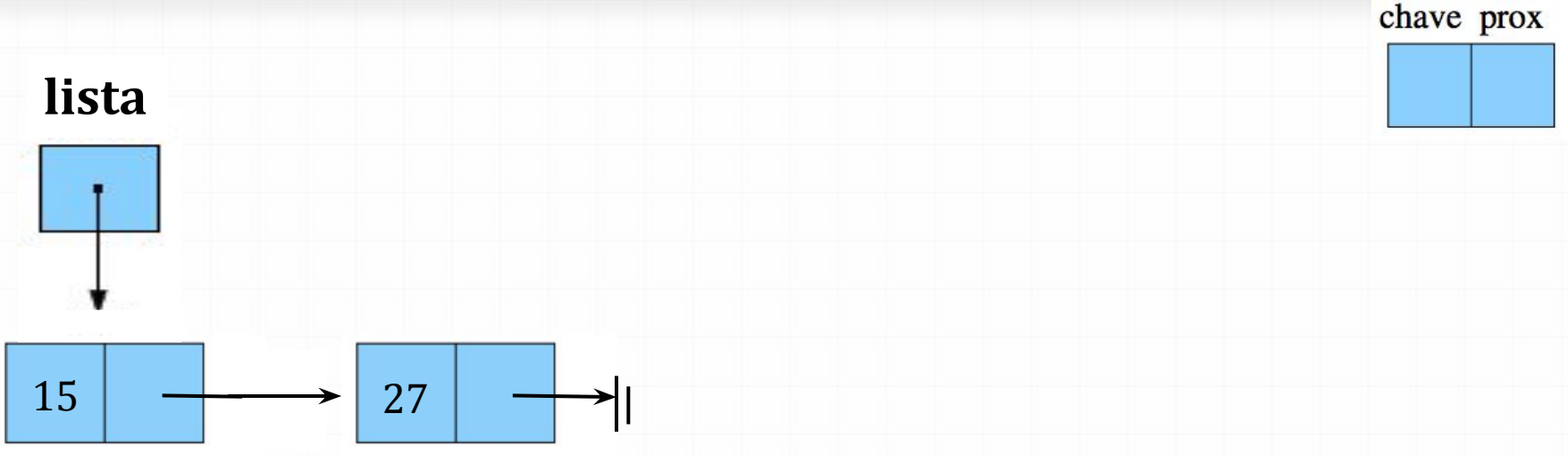
novo

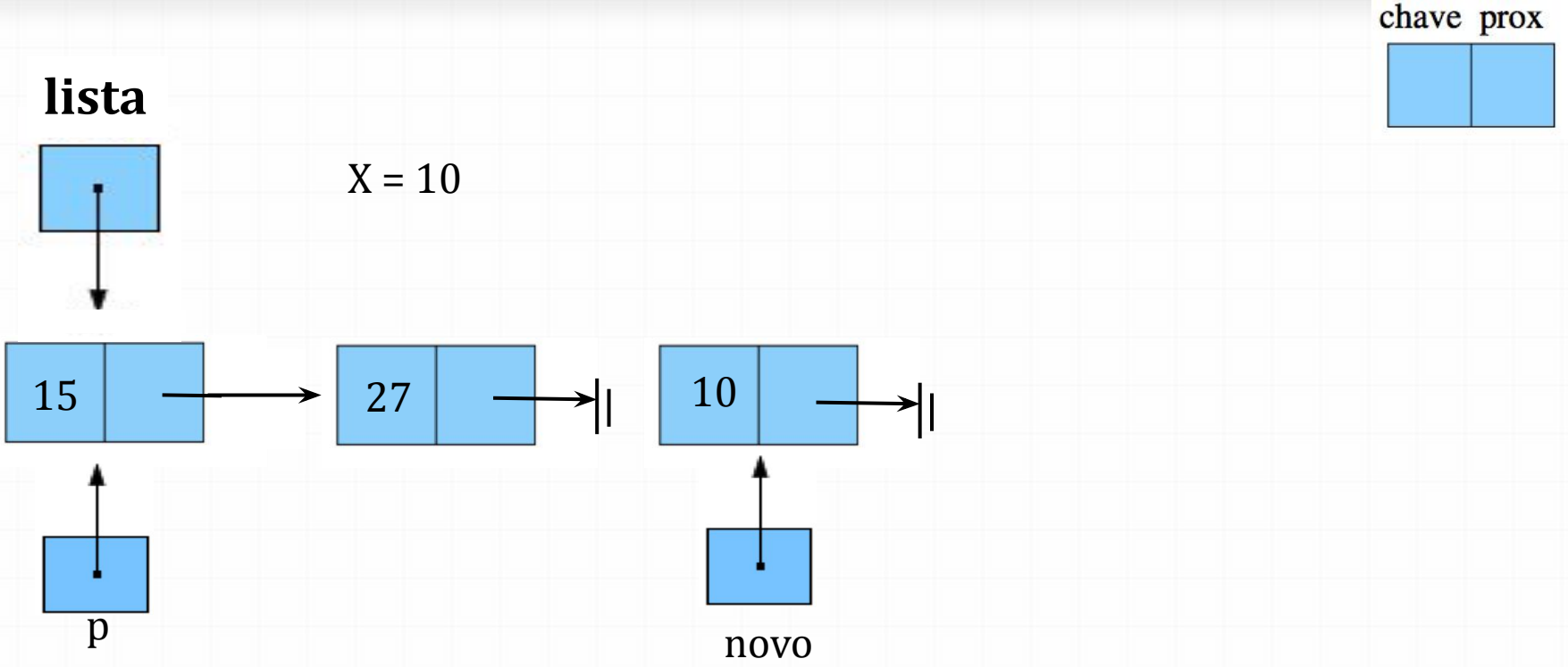

lista
15

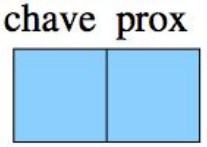

chave prox


chave prox

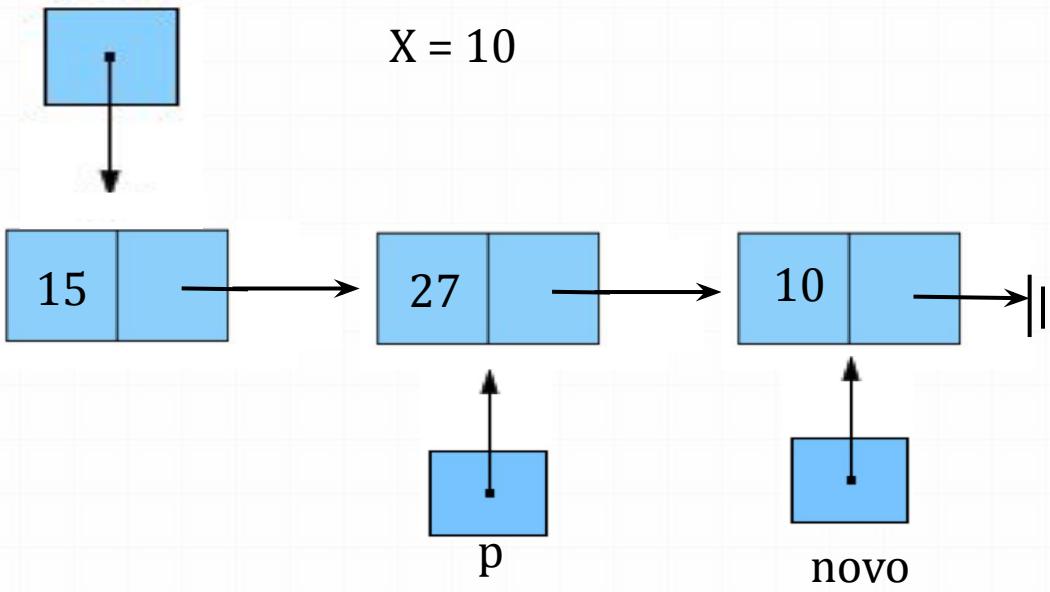

lista
X = 27
15
27
novo

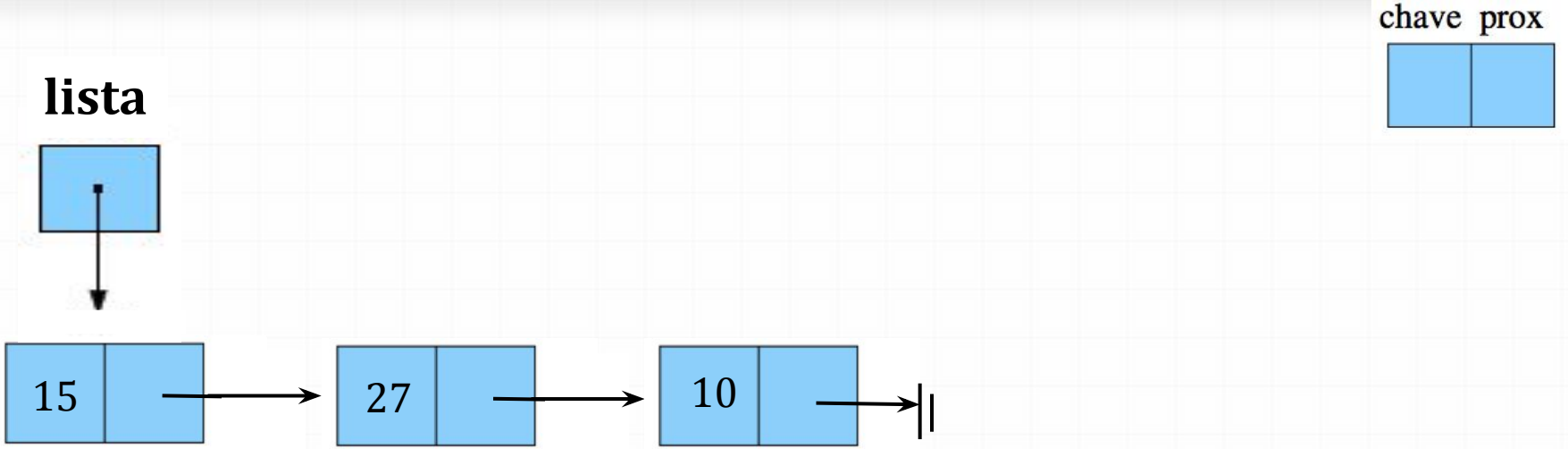

chave prox
lista
15
27

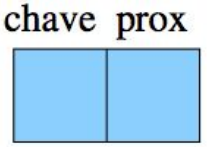

chave prox
lista
X = 10
15
27
10
p
novo



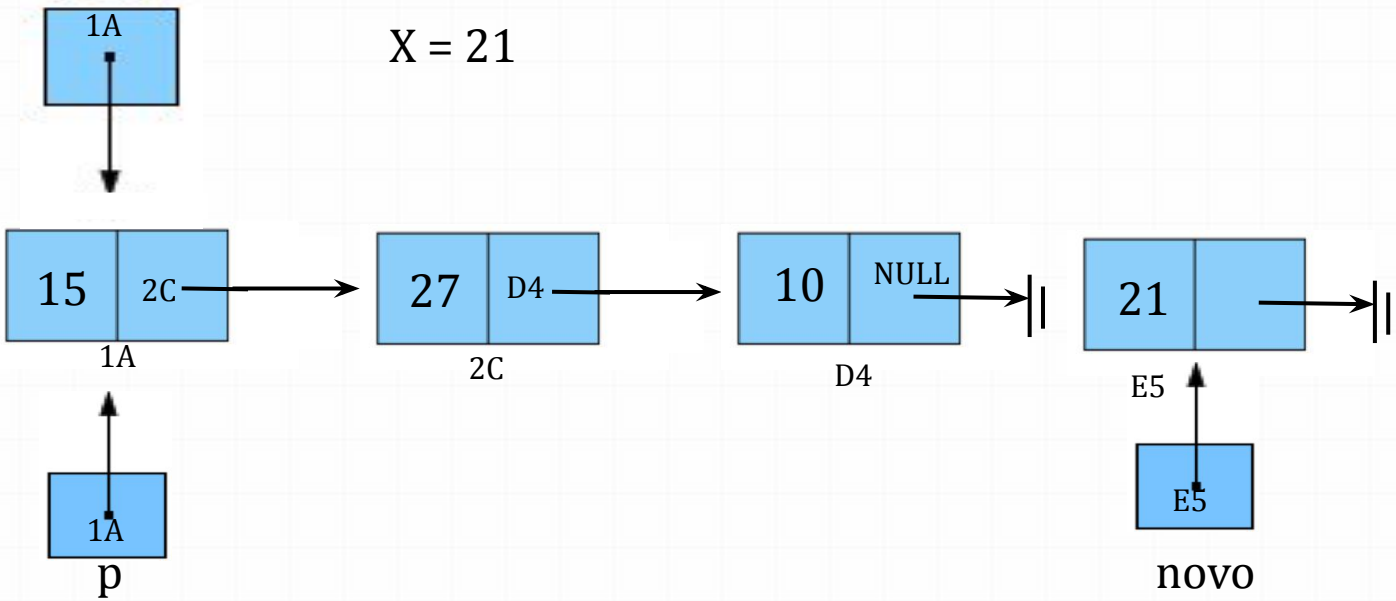
lista




chave prox
lista
15
27
10

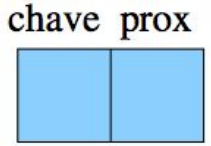


lista

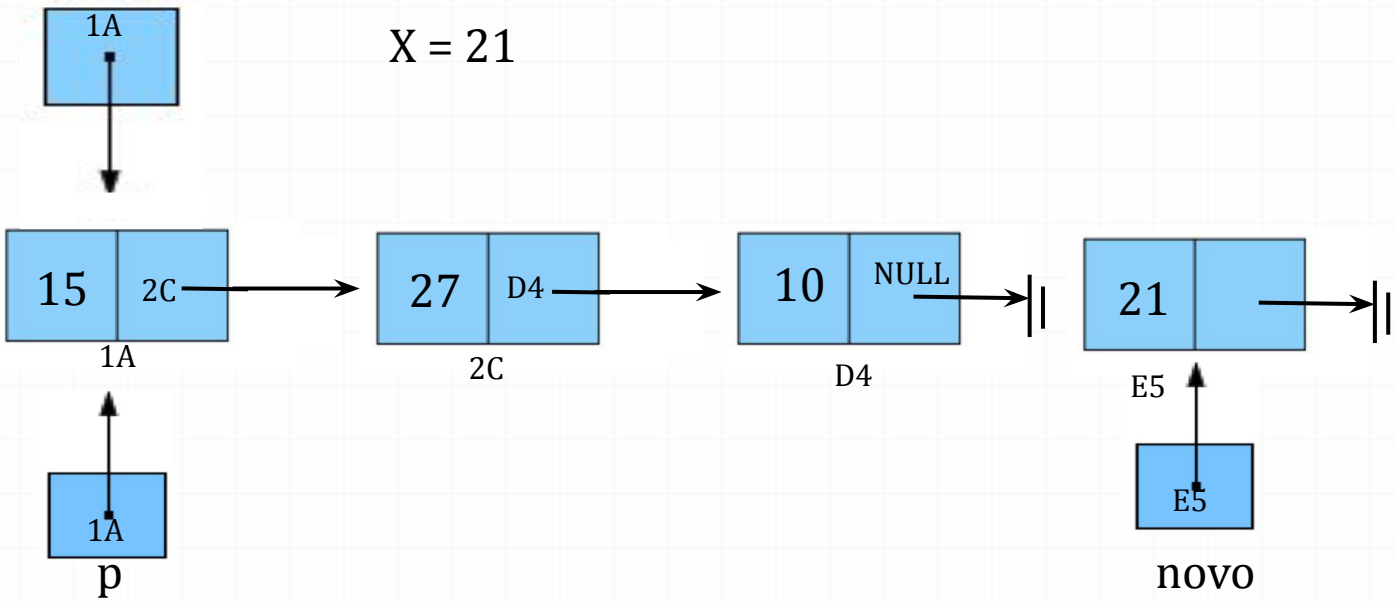



```
novo = (celula*) calloc (1, sizeof(celula));
novo->valor = X;
```

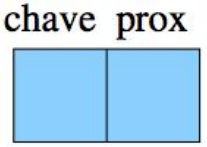
```
p = lista;
```



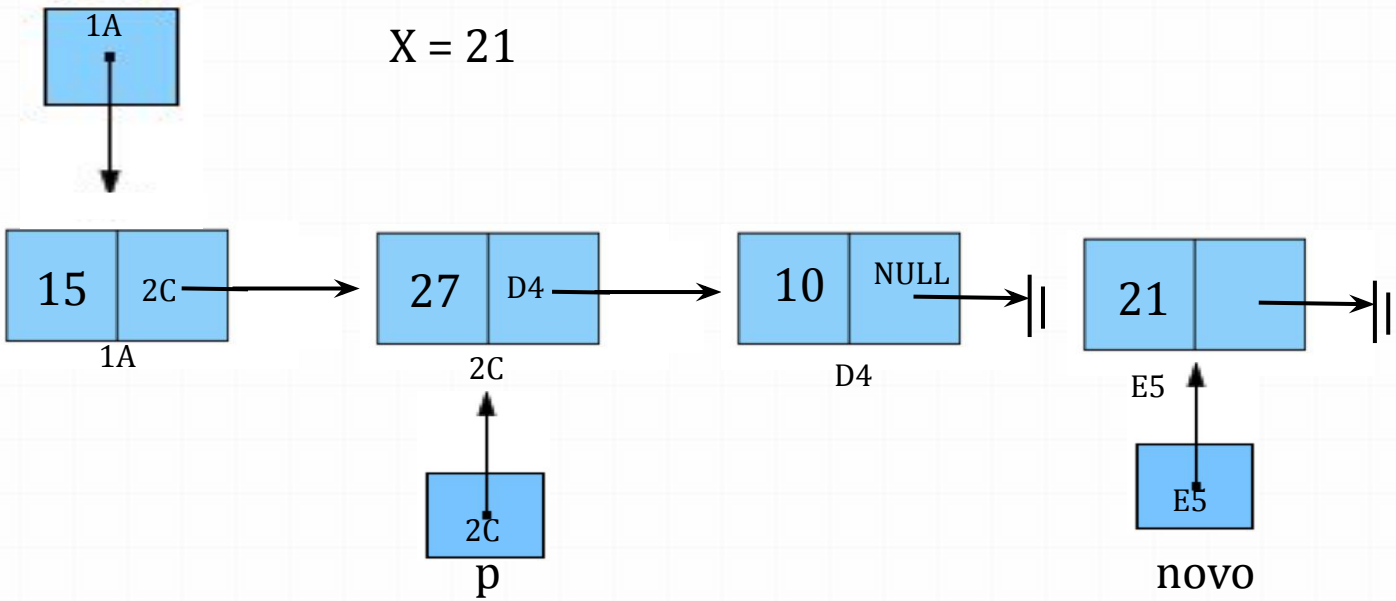
lista



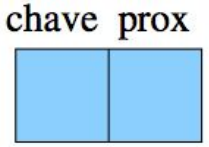

```
novo = (celula*) calloc (1, sizeof(celula));
novo->valor = X;

p = lista;
while(p->prox != NULL)
        p = p->prox;
```



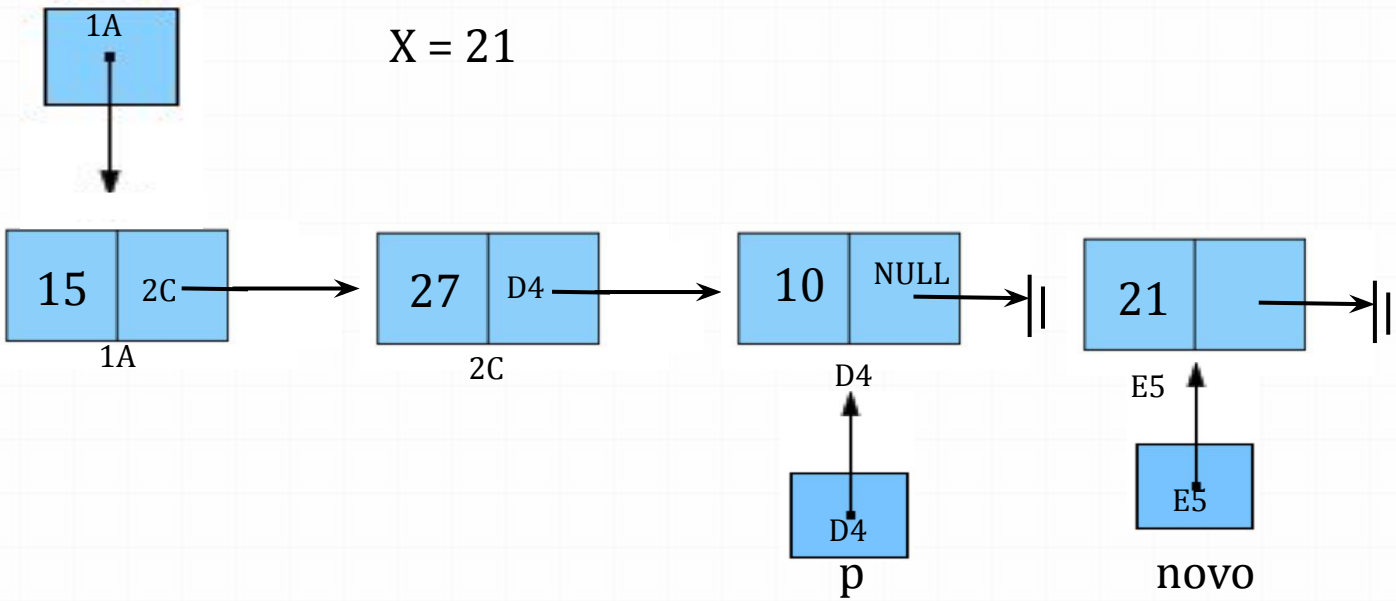
lista



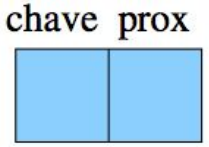

```
novo = (celula*) calloc (1, sizeof(celula));
novo->valor = X;

p = lista;
while(p->prox != NULL)
        p = p->prox;
```



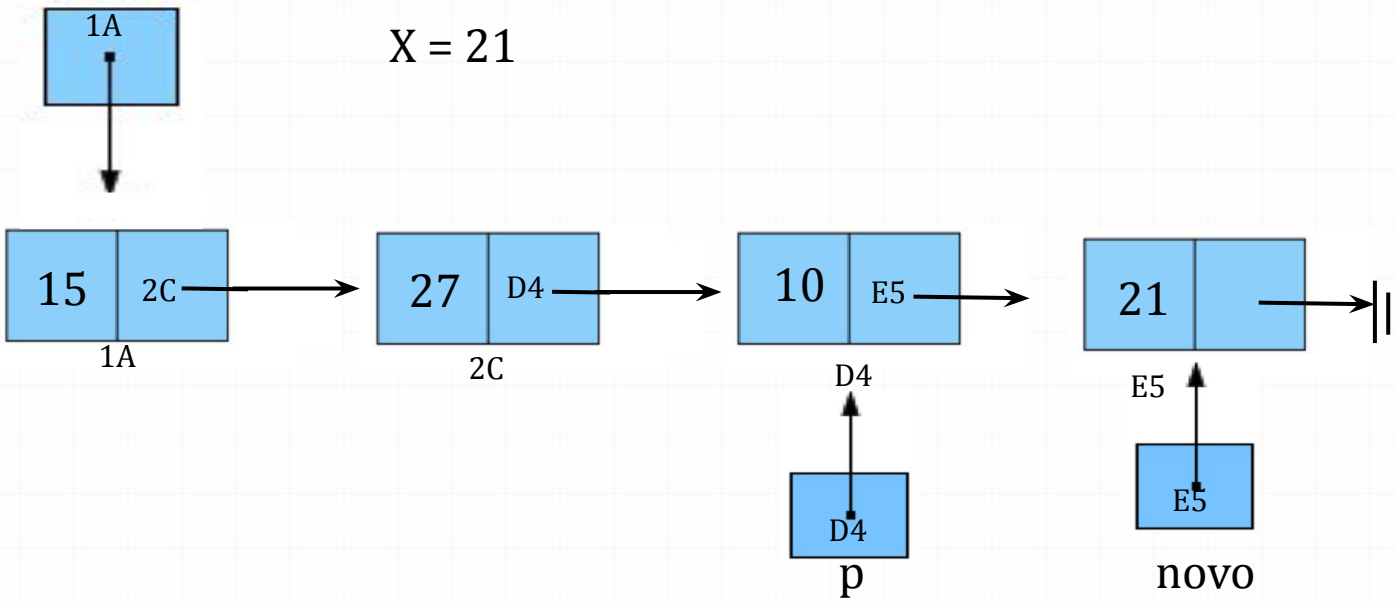
lista



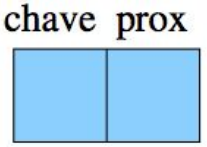

```
novo = (celula*) calloc (1, sizeof(celula));
novo->valor = X;

p = lista;
while(p->prox != NULL)
        p = p->prox;
```

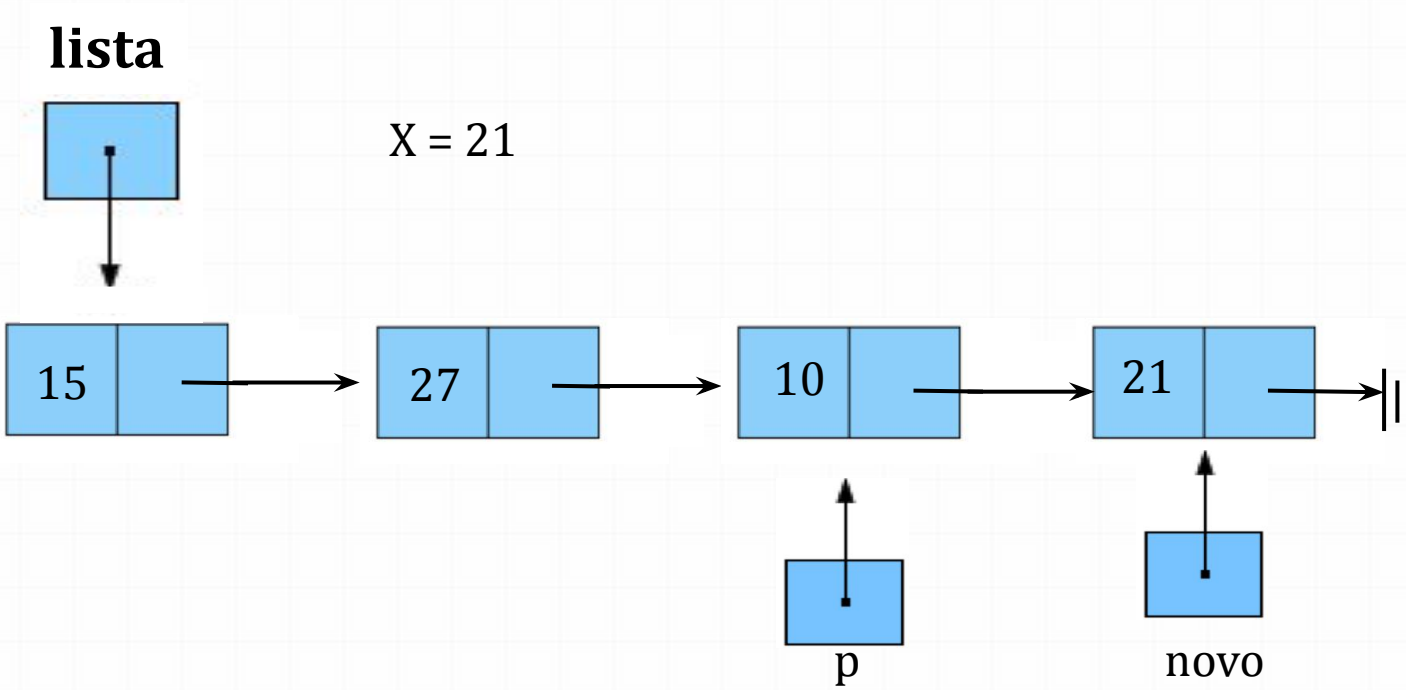


lista

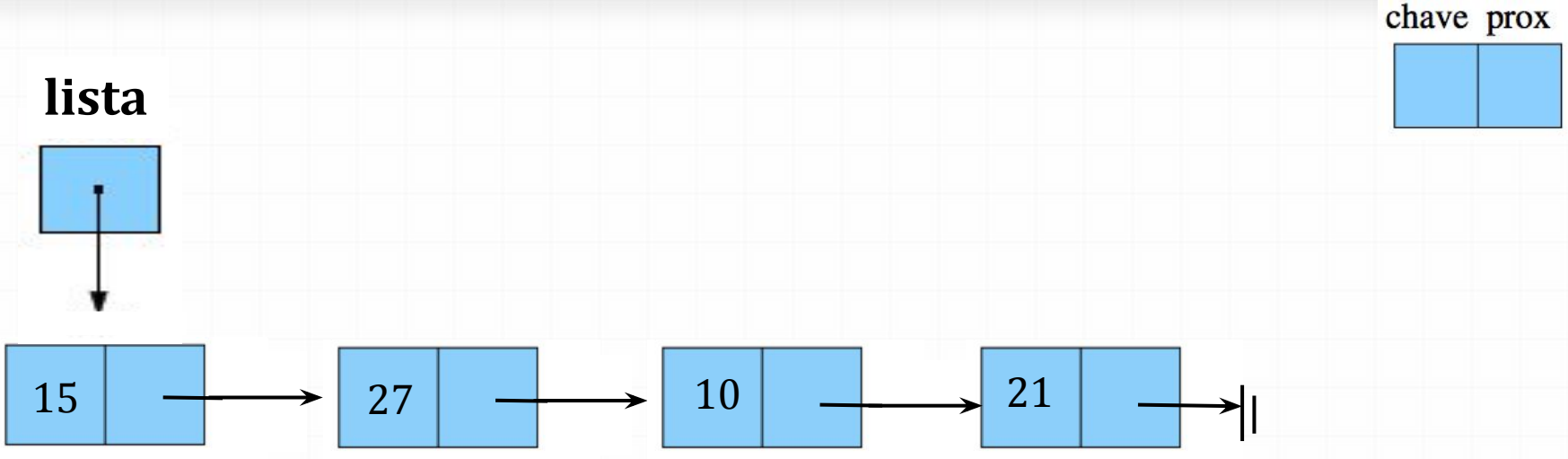



```
novo = (celula*) calloc (1, sizeof(celula));
novo->valor = X;
p = lista;
while(p->prox != NULL)
        p = p->prox;

p->prox = novo;
```






```
p = lst;
while(p->prox != NULL)
        p = p->prox;

p->prox = novo;
```


chave prox
lista
15
27
10
21

Protótipo da função (parâmetros: o valor a ser inserido e o ponteiro para a lista passado por referência).

```
void inserir_fim(int, celula*&);
```

Chamada da função inserir_fim, por exemplo:

```
/*dada uma lista simplesmente encadeada  'lista' e um dado na variável 'numero'*/
int main ( ){
    celula *lista=NULL;
    int numero;
    …
        inserir_fim(numero, lista);
    …
}
```

```
void inserir_fim(int x, celula*&lst)
{
    celula *novo, *p;

    novo = (celula*) malloc(sizeof(celula));
    novo->prox = NULL;
/*  novo = (celula*) calloc(1, sizeof(celula));*/
    novo->chave = x;
    if(lst == NULL)   /*lista esta vazia*/
        lst = novo;
    else{
        p = lst;
        while(p->prox != NULL)
            p = p->prox;

        p->prox = novo;
    }
```

lst (lista)

lst é um Apelido Para lista

OUTRA VERSÃO: Protótipo da função (parâmetros: o valor a ser inserido e o ponteiro para a lista passado por cópia).

celula* inserir_fim(int, celula*);

Chamada da função inserir_fim, por exemplo:

```
/*dada uma lista simplesmente encadeada  'lista' e um dado na variável 'numero'*/
int main ( ){
    celula *lista=NULL;
    int numero;
    …
        lista = inserir_fim(numero, lista);
    …
}
```

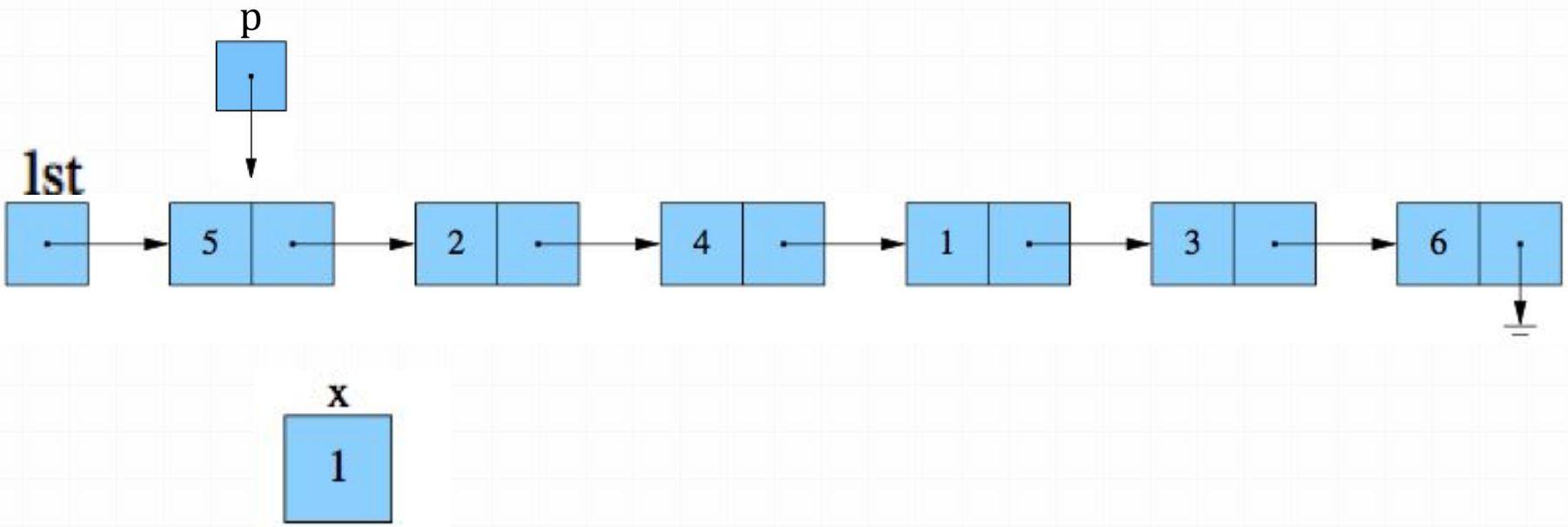
```
celula* inserir_fim(int x, celula *lst)
{
      celula *novo, *p;

      novo = (celula*) malloc(sizeof(celula));
      novo->prox = NULL;
/*    novo = (celula*) calloc(1, sizeof(celula));*/
      novo->chave = x;
      if(lst == NULL)   /*lista esta vazia*/
            lst = novo;
      else{
            p = lst;
            while(p->prox != NULL)
                  p = p->prox;

            p->prox = novo;
      }
      return lst;
}
```

lst recebeu Uma cópia de lista

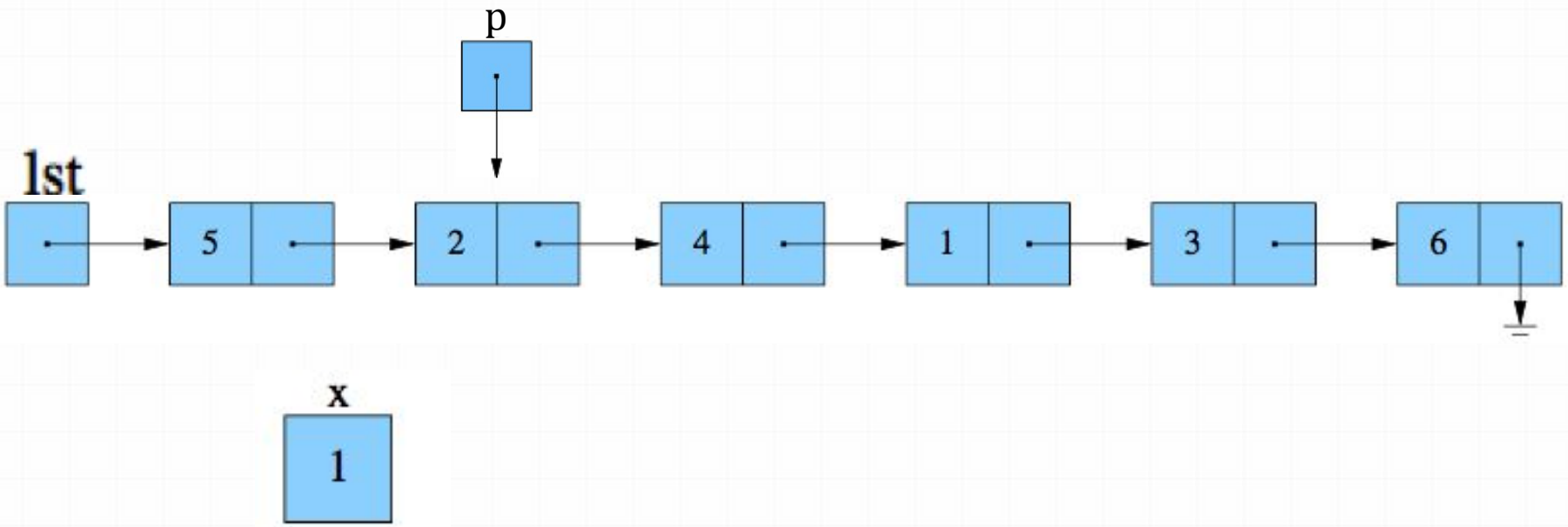
Operações sobre listas lineares sem cabeça

- Remoção de um elemento qualquer **x** da lista:
  - primeiro fazer uma busca por **x**
    - **Se for encontrado, remova-o.**

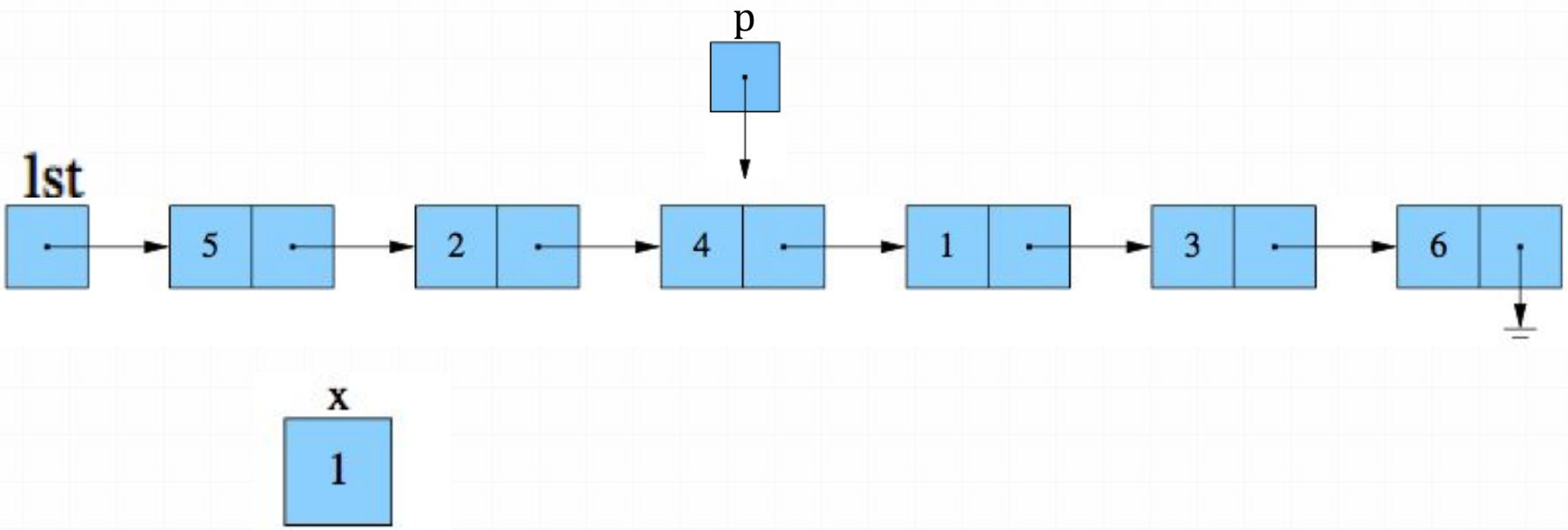
Operações sobre listas lineares sem cabeça



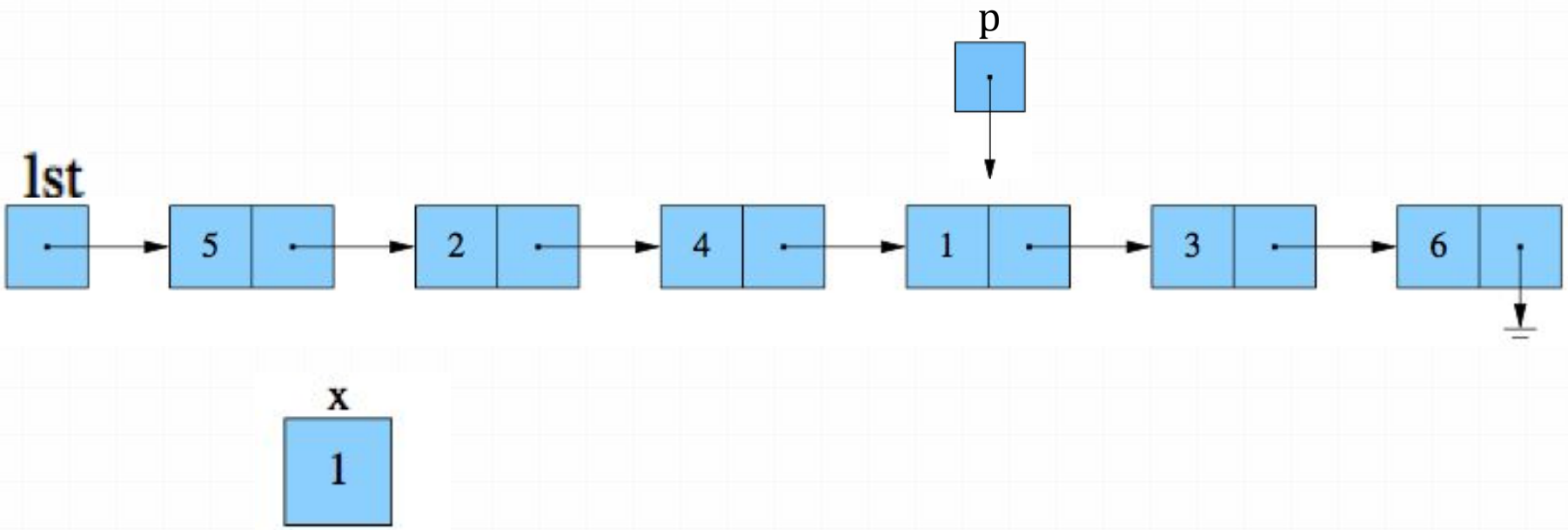
Operações sobre listas lineares sem cabeça



Operações sobre listas lineares sem cabeça

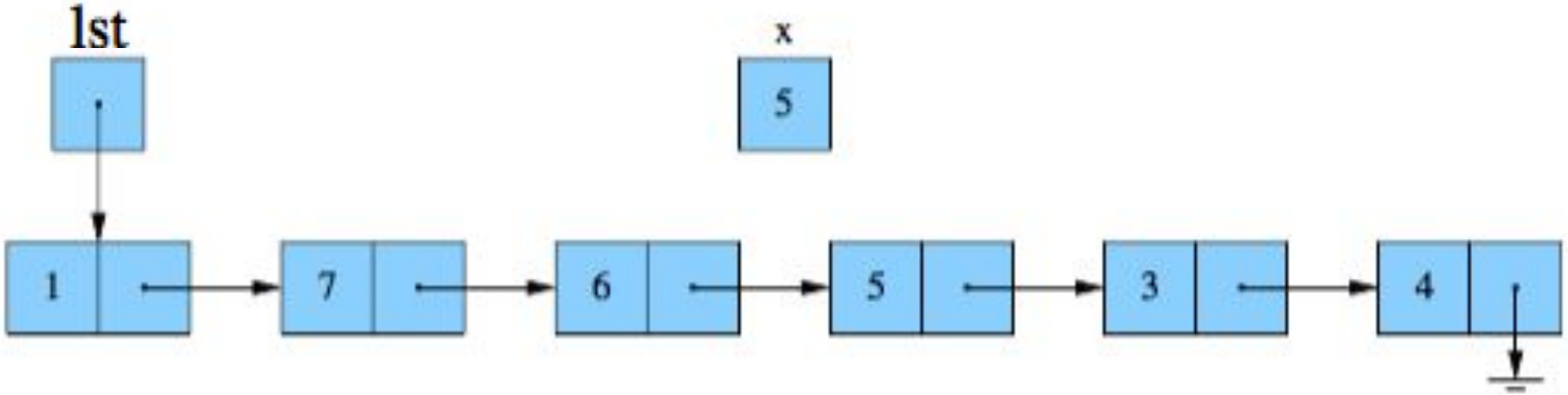


Operações sobre listas lineares sem cabeça

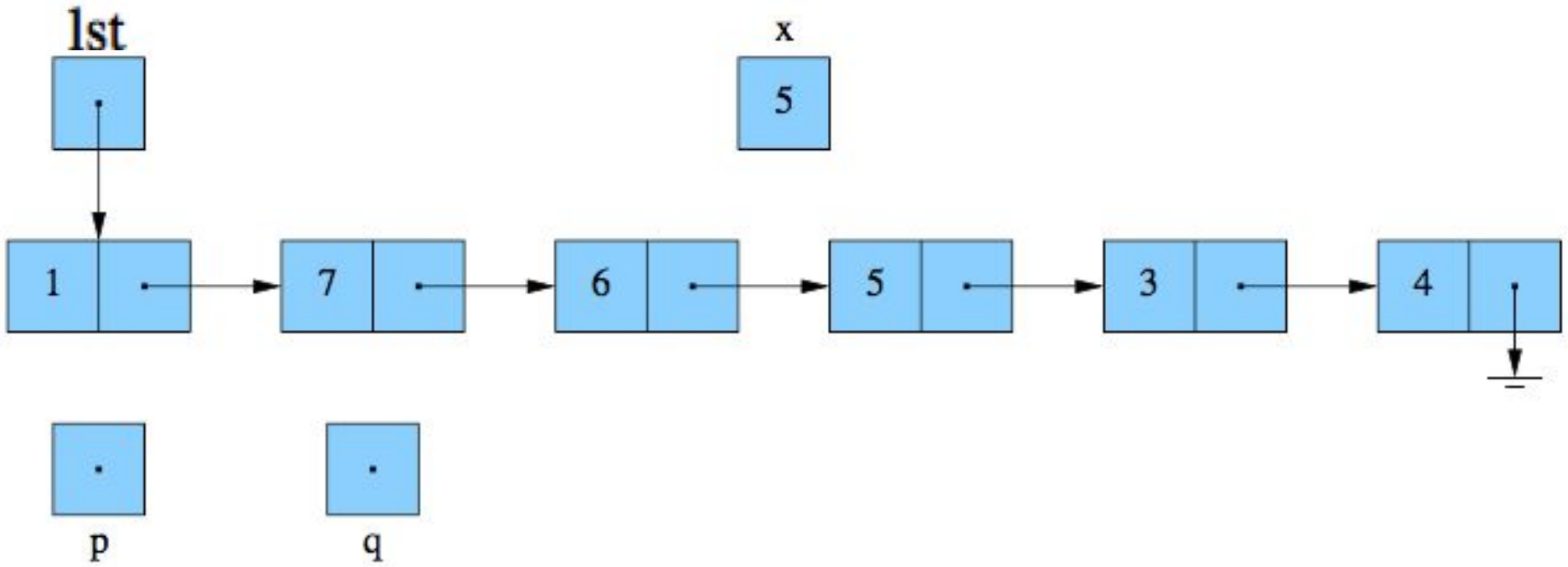


Operações sobre listas sem cabeça

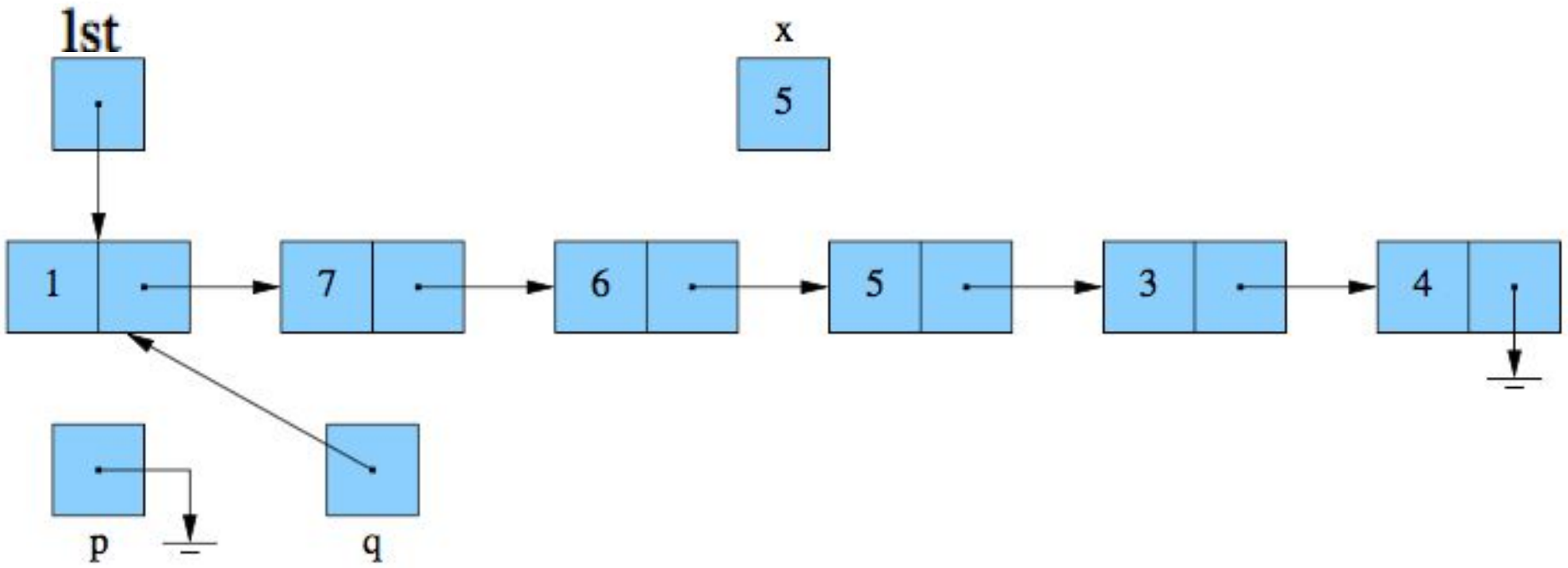
- Remoção de um elemento **x** da lista:



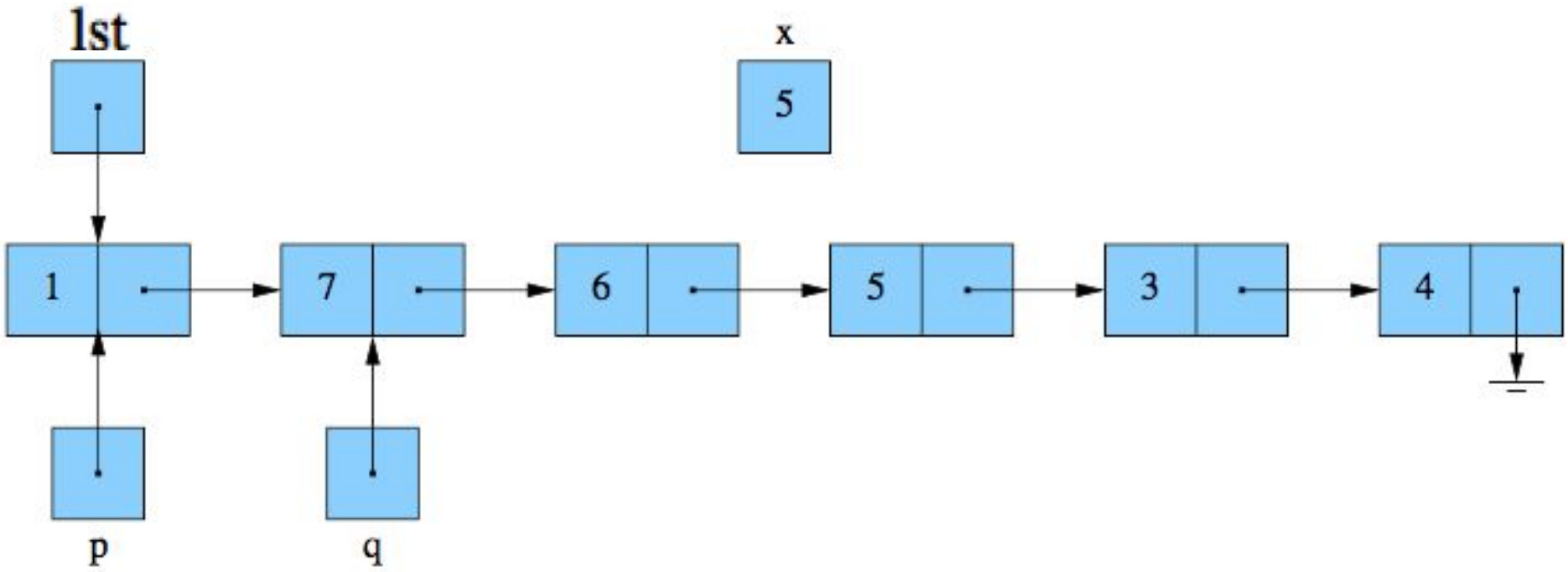
Operação de remoção



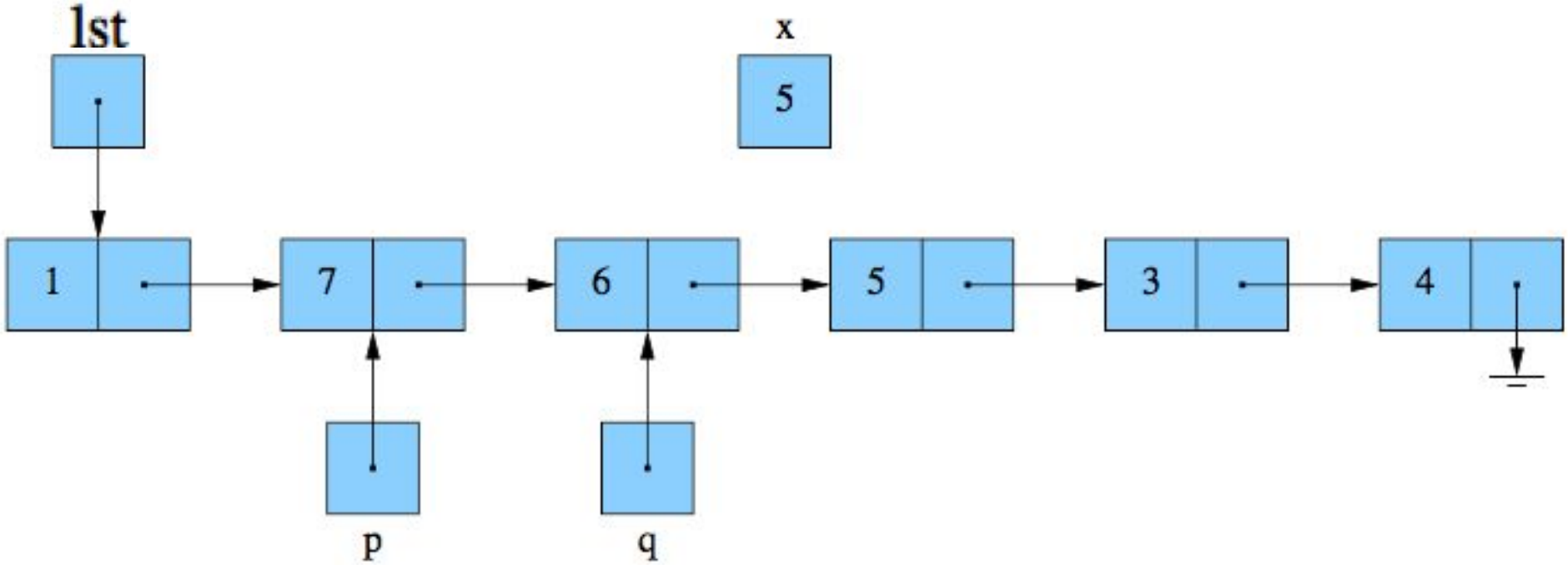
Operação de remoção



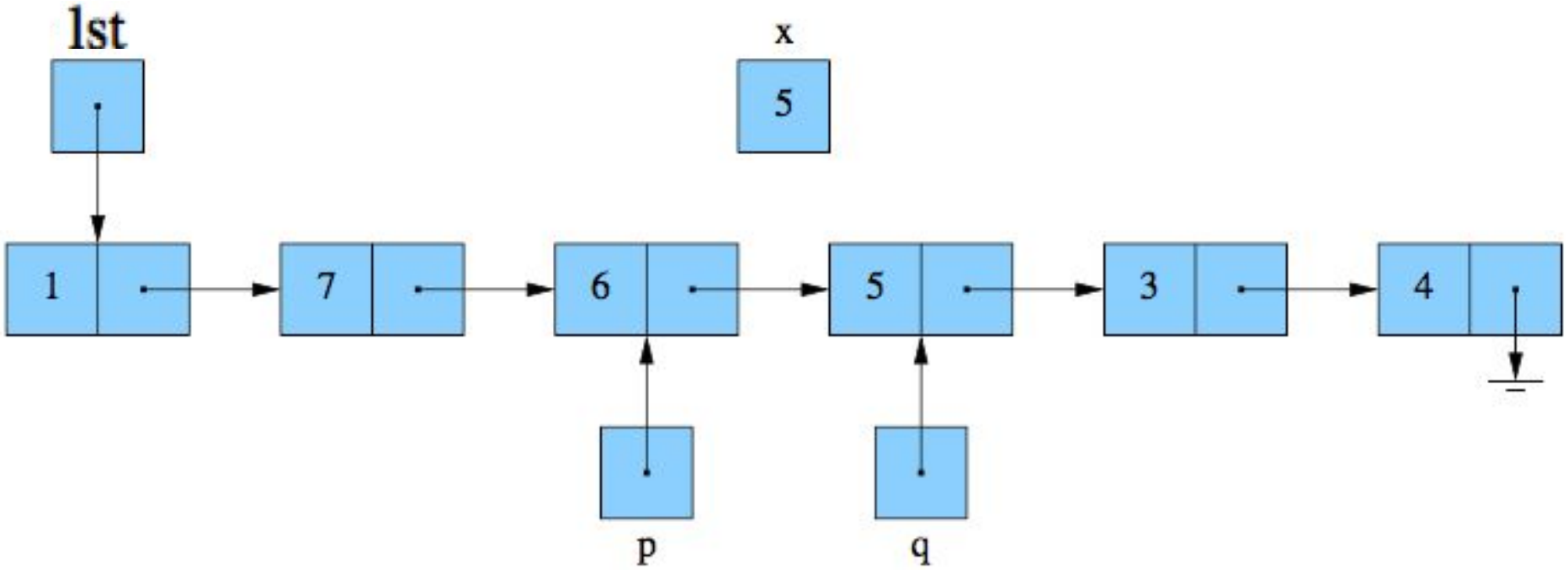
Operação de remoção



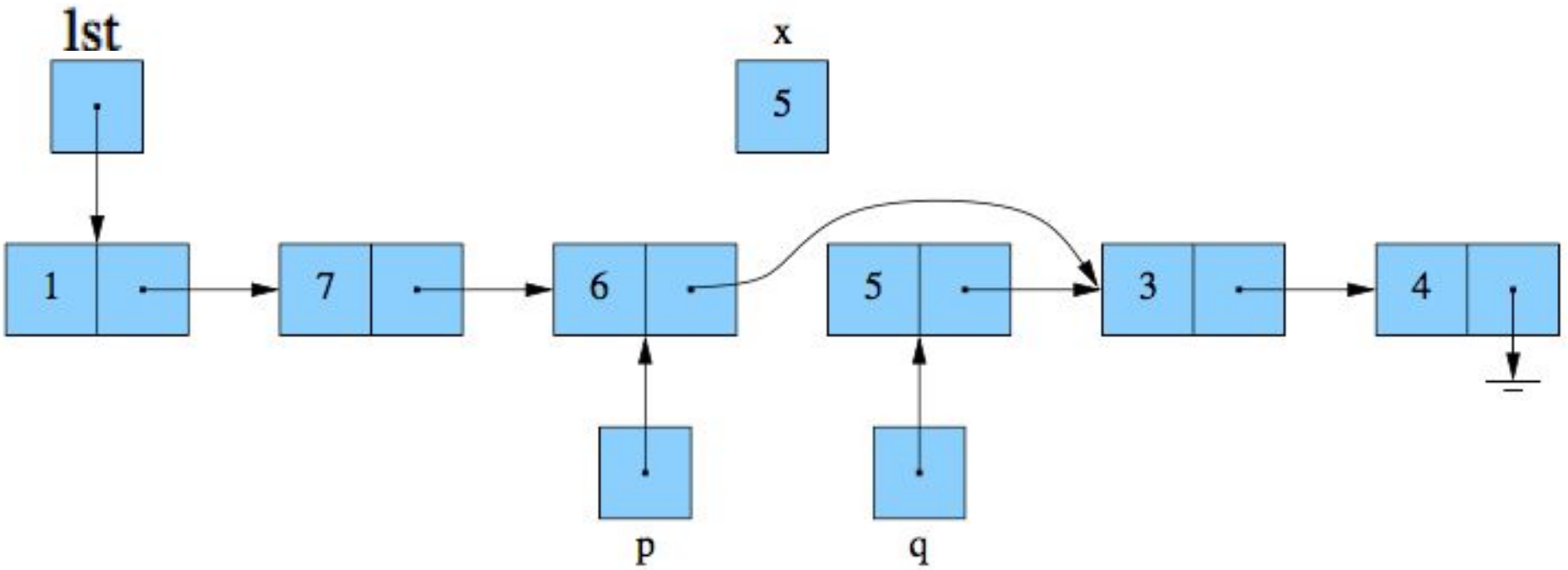
Operação de remoção



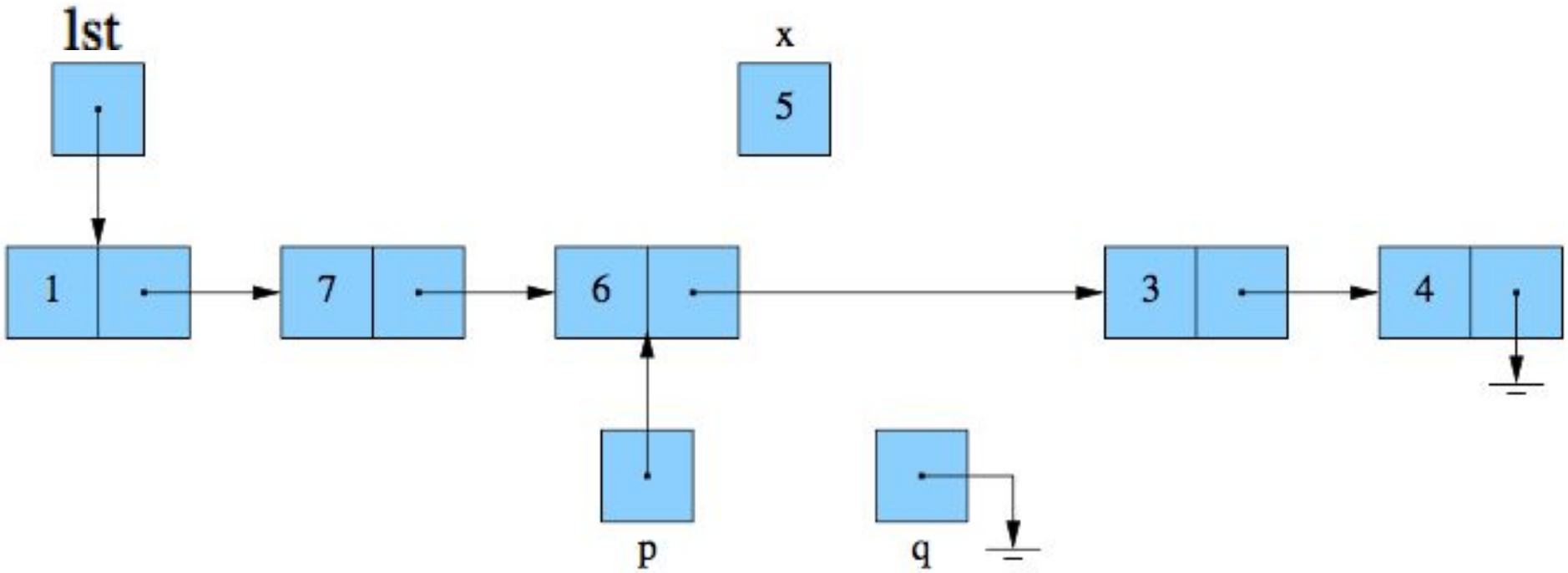
Operação de remoção



Operação de remoção



Operação de remoção



Chamada à função de remoção

- Seja **lista** uma lista linear sem cabeça e ***x*** uma chave a ser removida de lista
- Se temos uma função com o protótipo

void busca_remove_S(int x, celula*&lst)

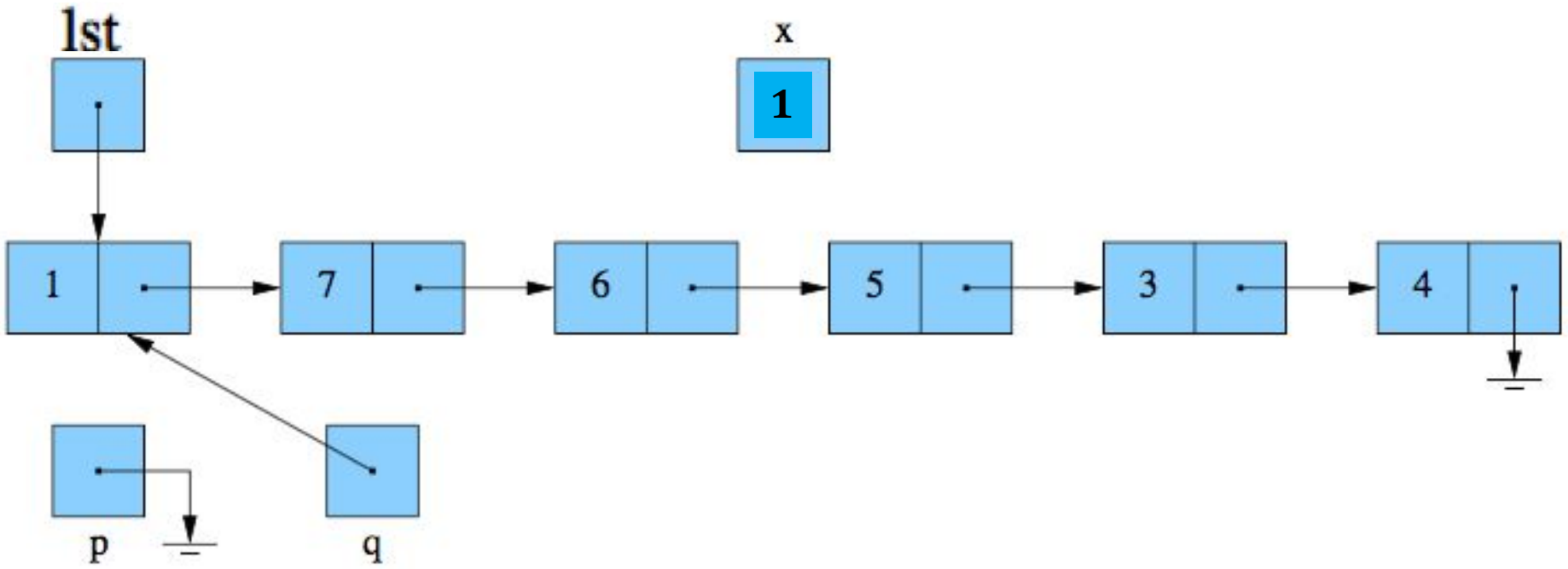
- A chamada seria: (**lista** é o ponteiro para primeiro nó)

celula *lista ;

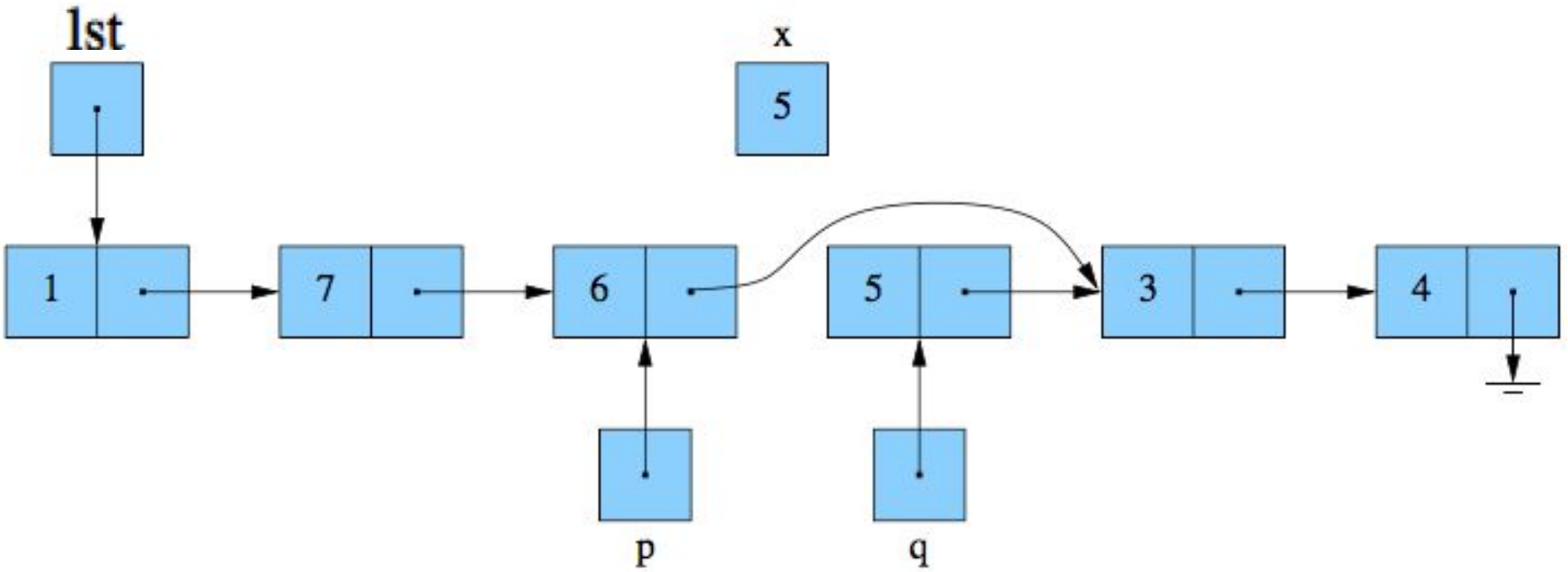
busca_remove_S(x, lista)

```
/* Recebe um número inteiro x e uma lista encadeada lst e remove da lista a
primeira célula que contiver x, se tal célula existir */
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;
                        free(q);
                }
                else {
                        lst = q->prox;
                        free(q);
                }
}
```

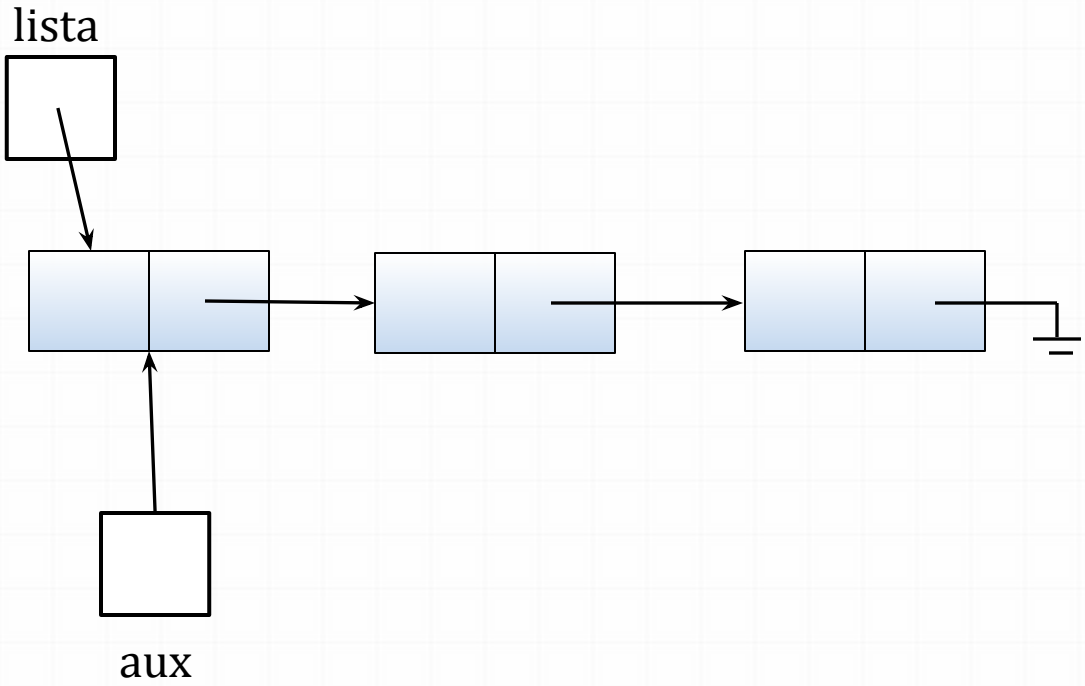
Operação de remoção



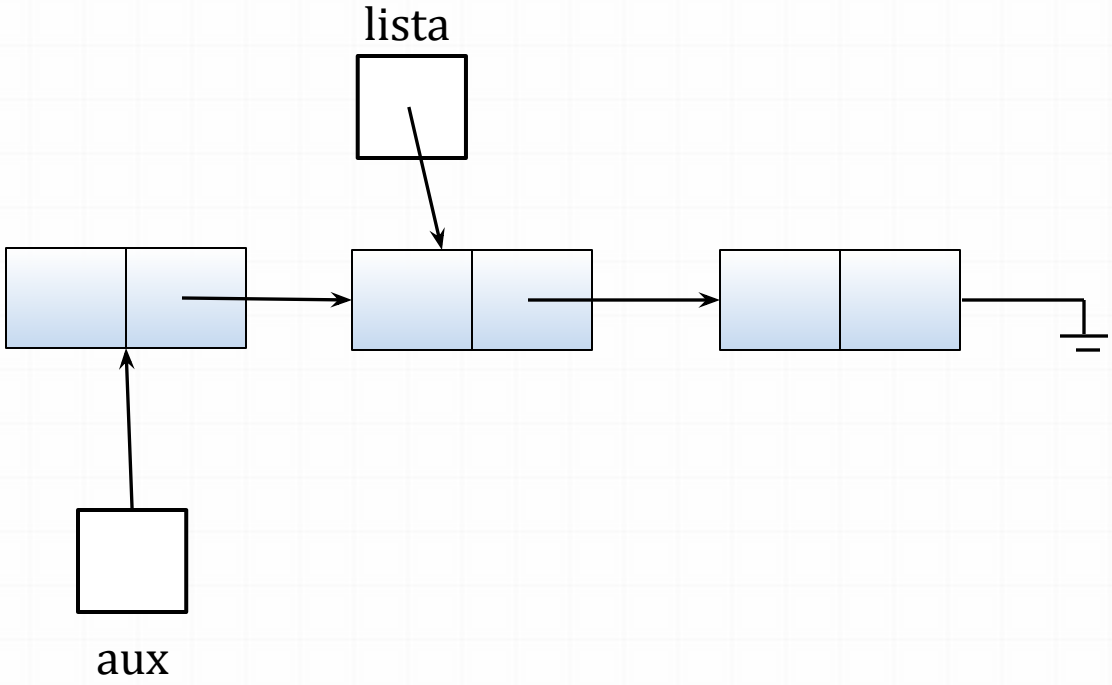
Operação de remoção



Apagando toda a lista

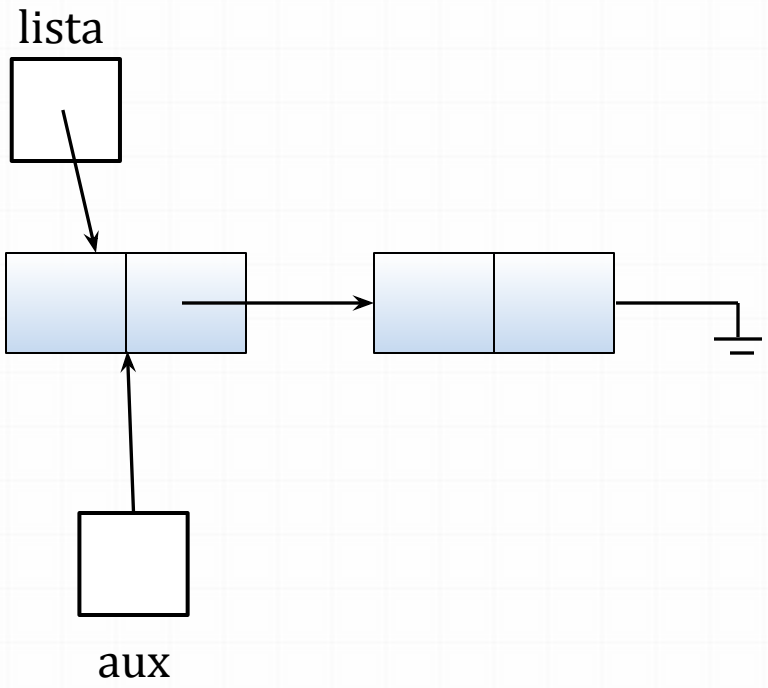


Apagando toda a lista

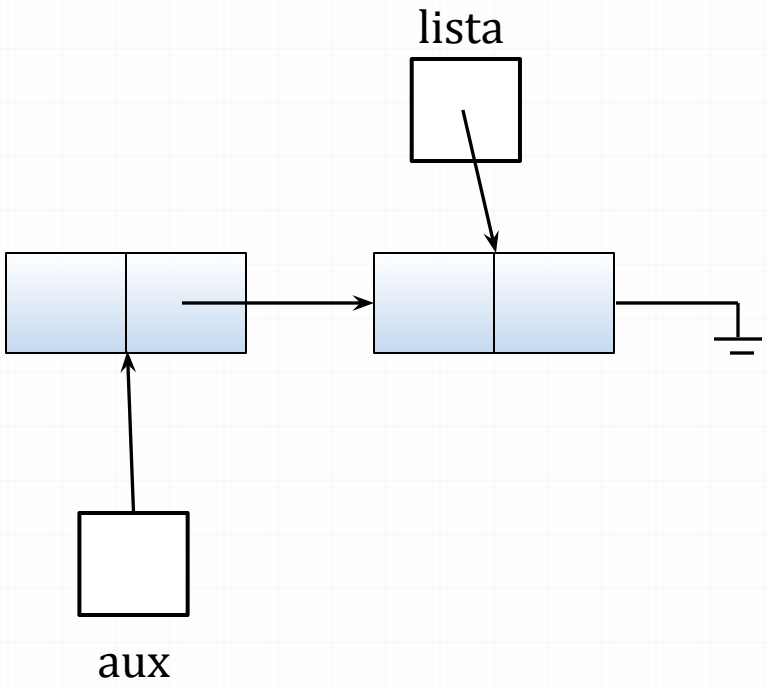


free(aux)

Apagando toda a lista

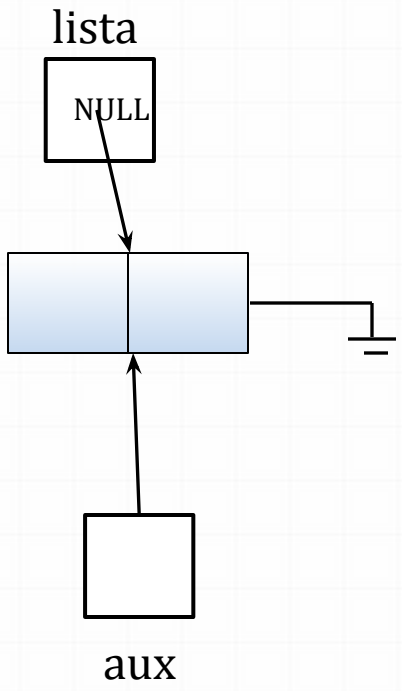


Apagando toda a lista



free(aux)

Apagando toda a lista



free(aux)

Apagando toda a lista