Lista Linear Duplamente Encadeada

Algoritmos e Programação II

(slides baseados na apostila do Prof Fábio Viduani)

Introdução

- lista linear especial
- às vezes precisamos manter um ponteiro para uma célula à frente e também para a célula anterior para realizar algumas operações sobre a lista
- **podemos precisar percorrer a lista linear nos dois sentidos**
- para solucionar esse problema, adicionamos um novo campo ponteiro nas células (nós) da lista, que aponta para a célula anterior da lista

Definição

- células de uma lista linear duplamente encadeada são ligadas por ponteiros que indicam o endereço da célula anterior e da próxima célula da lista

Definição

- Uma célula é definida como

C

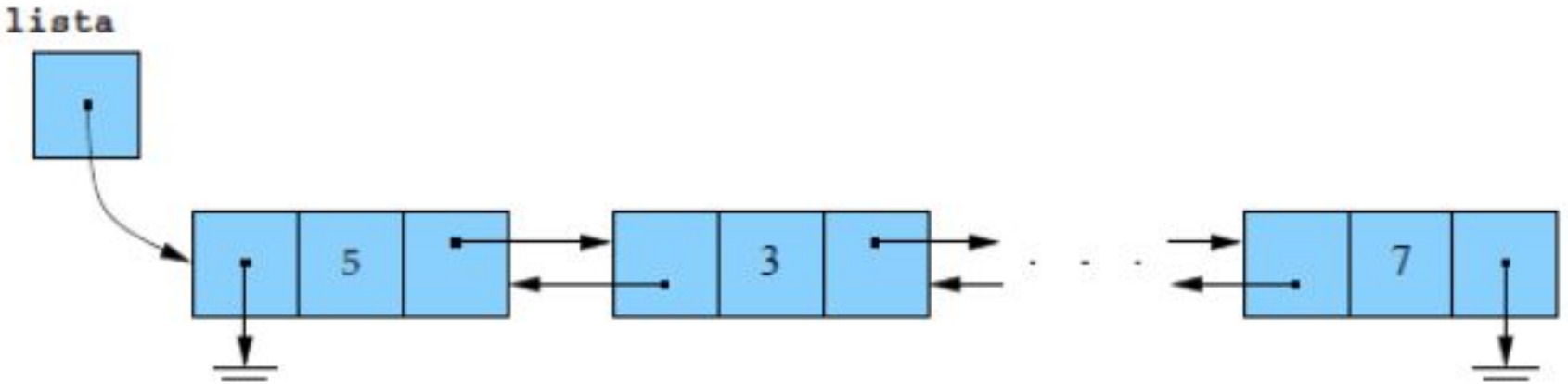
```
typedef struct cel {
    int chave;
    struct cel *ant;
    struct cel *prox;
} celula;
```

C++

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

Definição

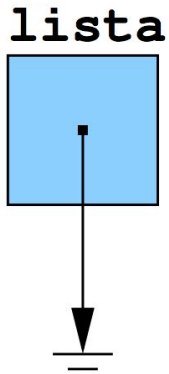
Um exemplo de uma lista linear duplamente encadeada



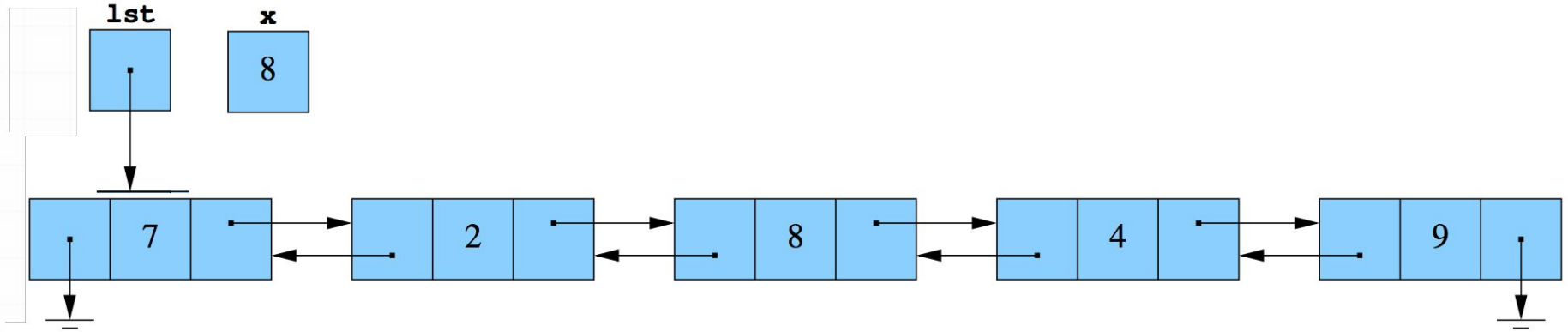
Definição

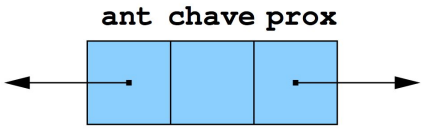
- **declaração e inicialização de uma lista linear duplamente encadeada sem cabeça**

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

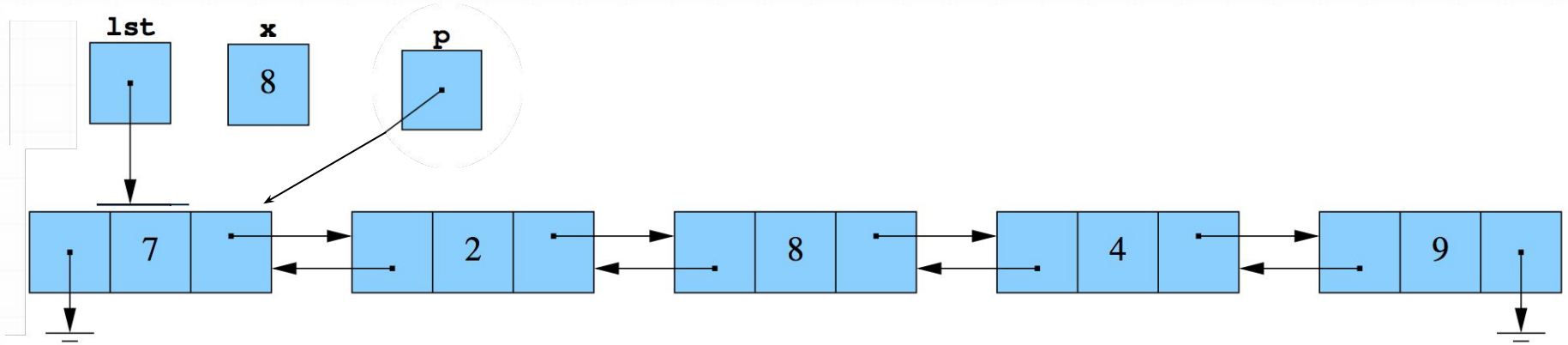


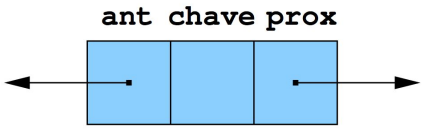
Operação de Busca



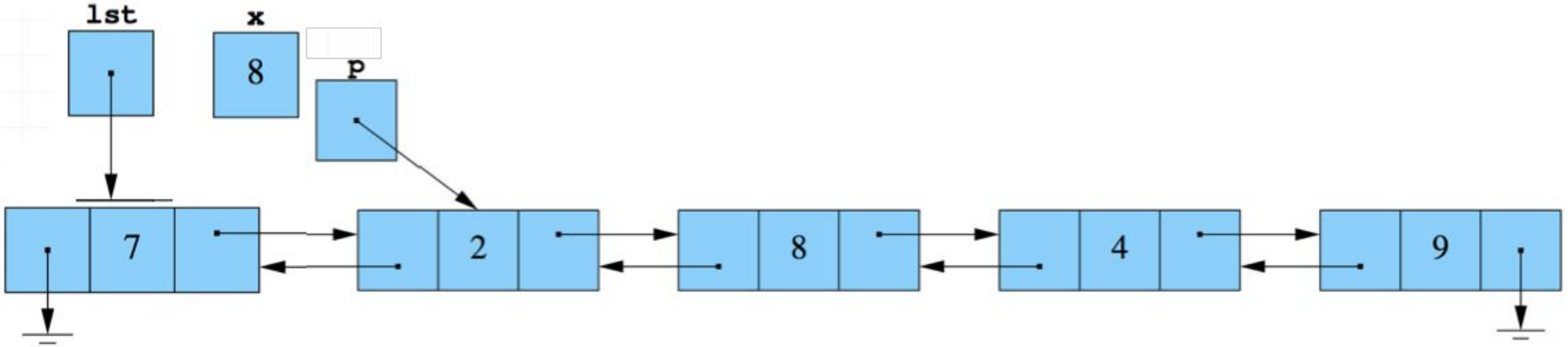


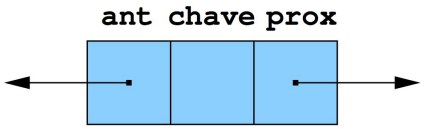
Operação de Busca



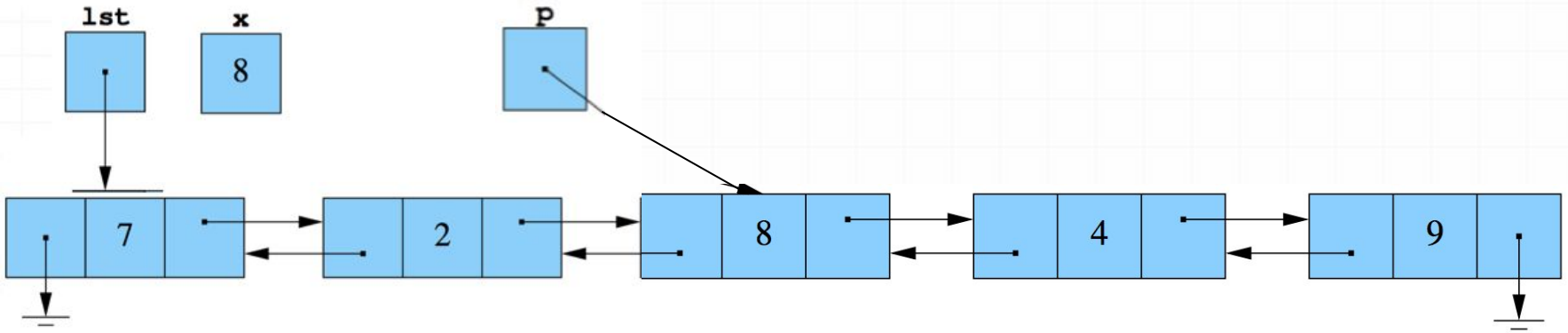


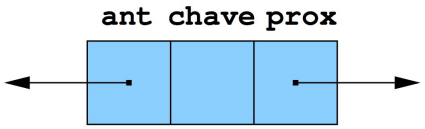
Operação de Busca





Operação de Busca





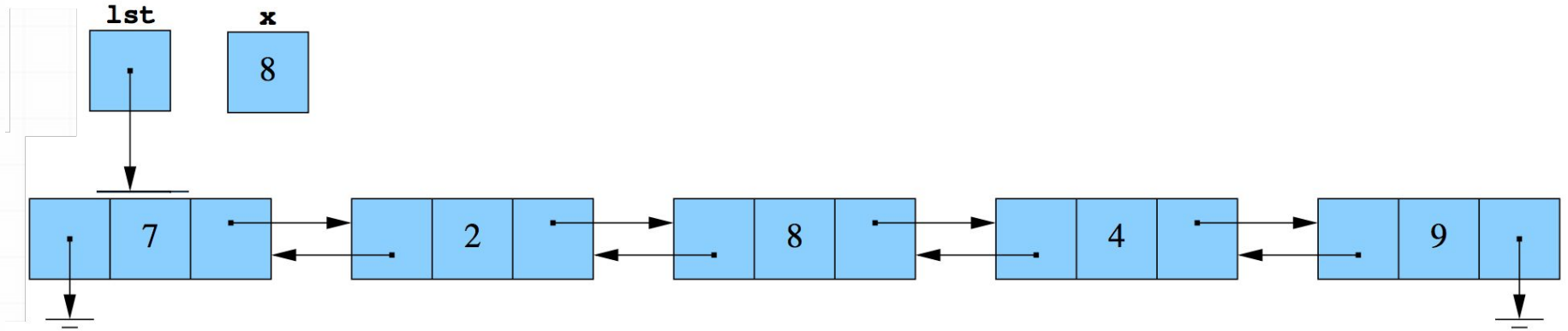
Operação de Busca

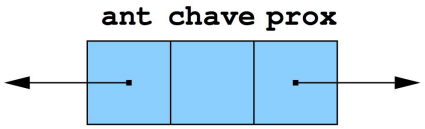
```
/* Recebe uma chave x e uma lista linear duplamente encadeada lst e devolve a
célula que contém x em lst ou NULL caso contrário */
celula *busca_dup(int x, celula *lst)
{
    celula *p;
    p = lst;
    while (p != NULL && p->chave != x)
        p = p->prox;
    return p;
}
```

Operação de Busca

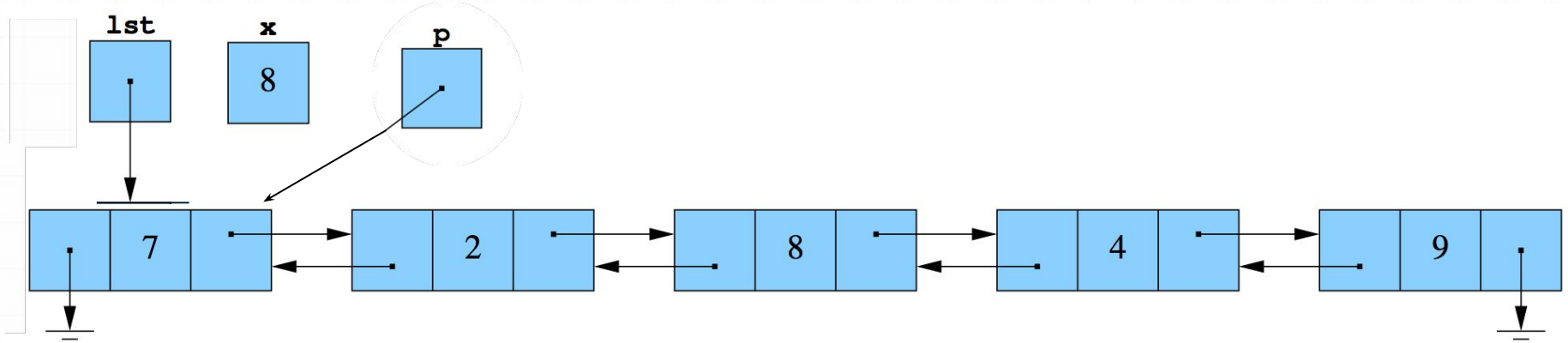
```
int main()
{
    celula *lista = NULL, *ptr;
    int x;
    .
    .
    .
    ptr = busca_dup( x, lista );
    if(ptr == NULL)
        printf("%d nao foi encontrado.\n", x);
    else
        printf("%d", ptr->chave);
    ...
}
```

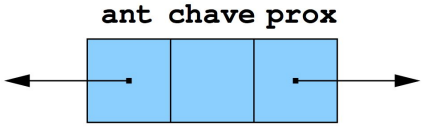
Busca seguida de remoção



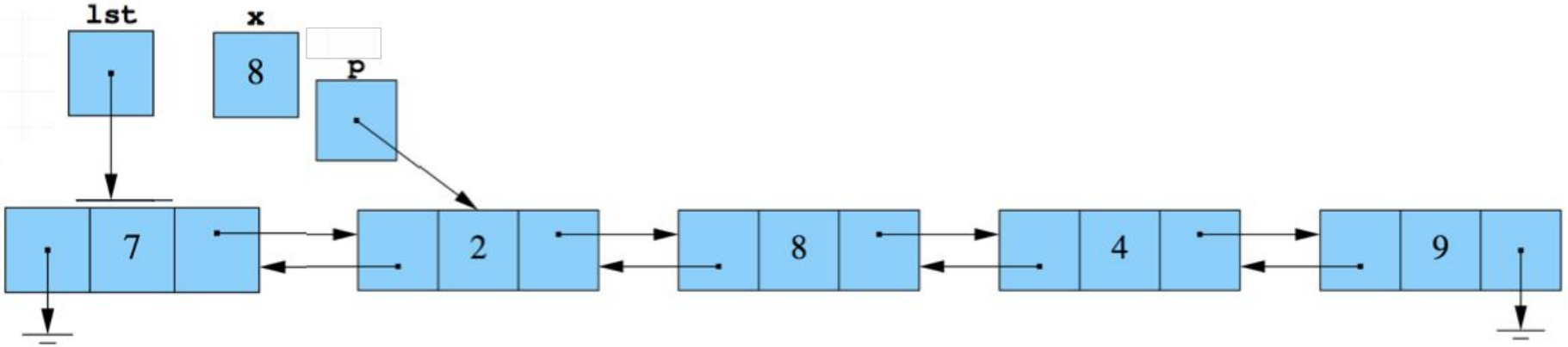


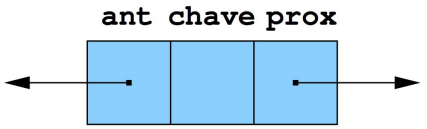
Busca seguida de remoção



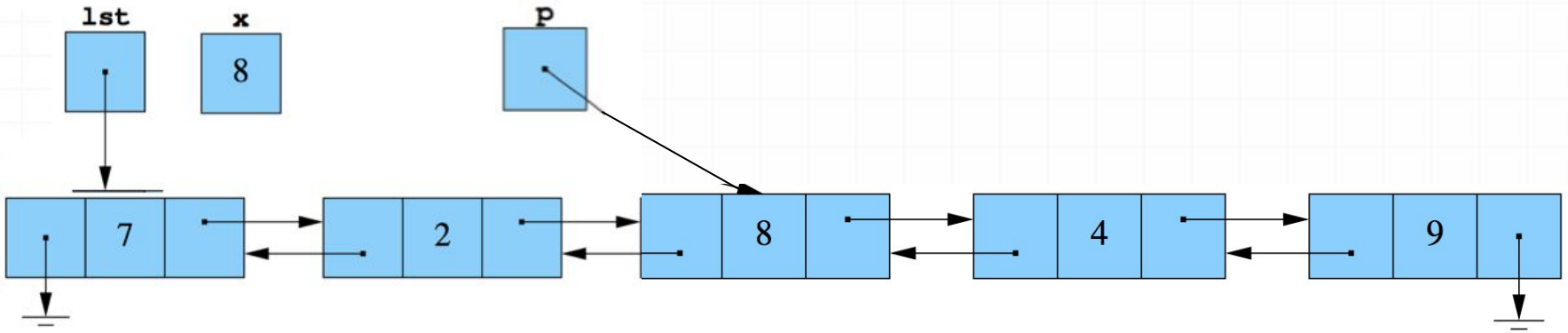


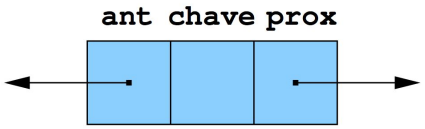
Busca seguida de remoção



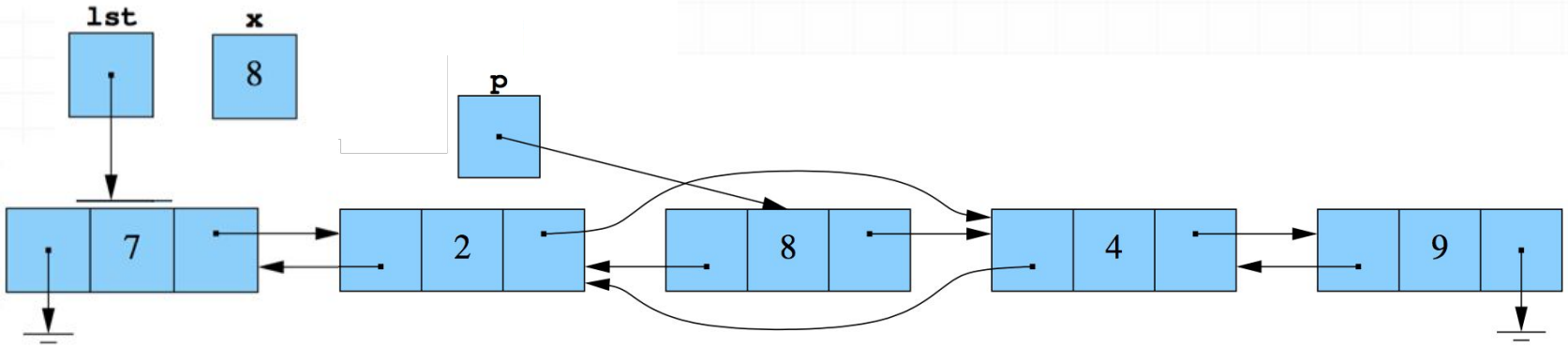


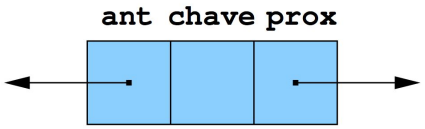
Busca seguida de remoção



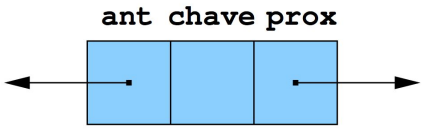


Busca seguida de remoção





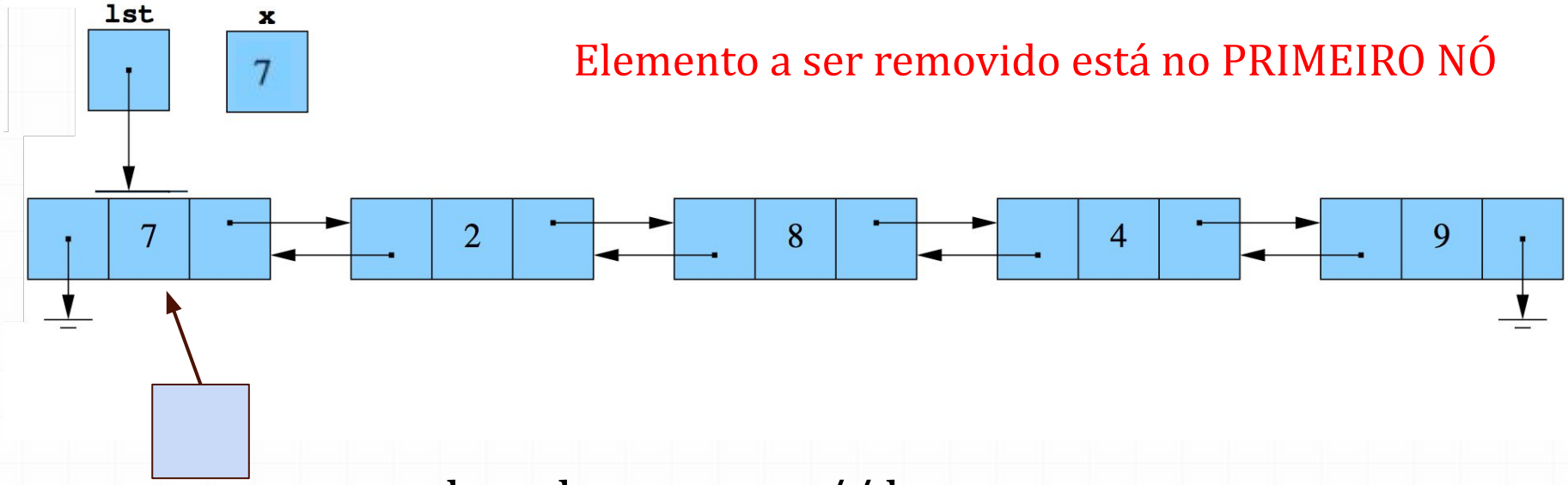
Busca seguida de remoção



Busca seguida de remoção

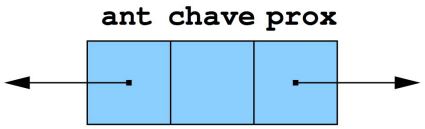
```
/* Recebe um número inteiro x e uma lista duplamente encadeada lst (por referência)
e remove da lista a primeira célula que contiver x, se tal célula existir */
void busca_remove_dup(int x, celula*& lst)
{
     celula *p;

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

    if(p == NULL)
        printf("\nNao encontrado");
    else{
        if(p->ant == NULL)   /*nó a remover é o primeiro*/
        {
            lst = lst->prox;
            if(lst != NULL)
                lst->ant = NULL;
        }
        else{
            p->ant->prox = p->prox;
            if(p->prox!=NULL)    /*caso p seja o último nó, seu prox é NULL*/
                p->prox->ant = p->ant;
        }
        free(p);
    }
}
```

Busca seguida de remoção

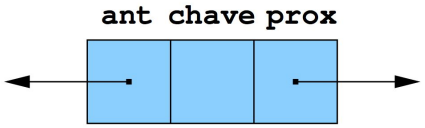


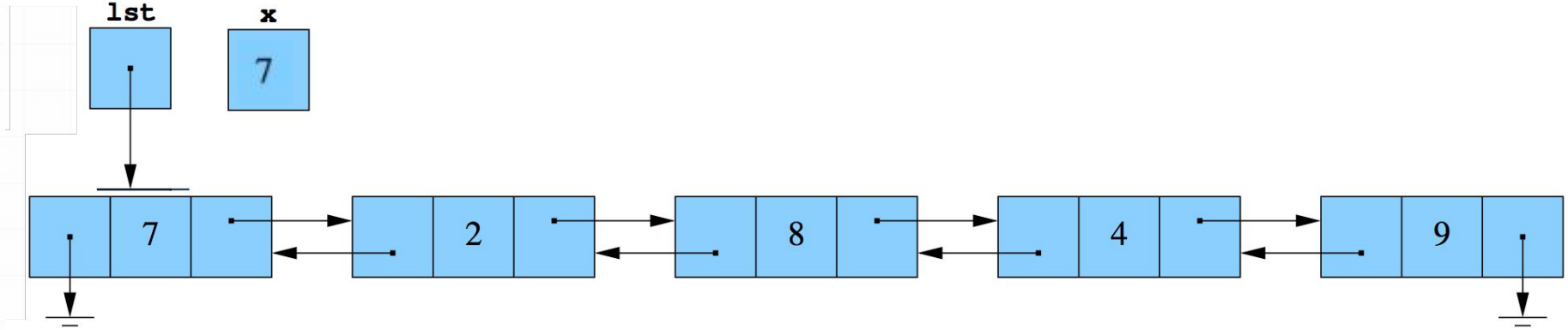
Elemento a ser removido está no PRIMEIRO NÓ

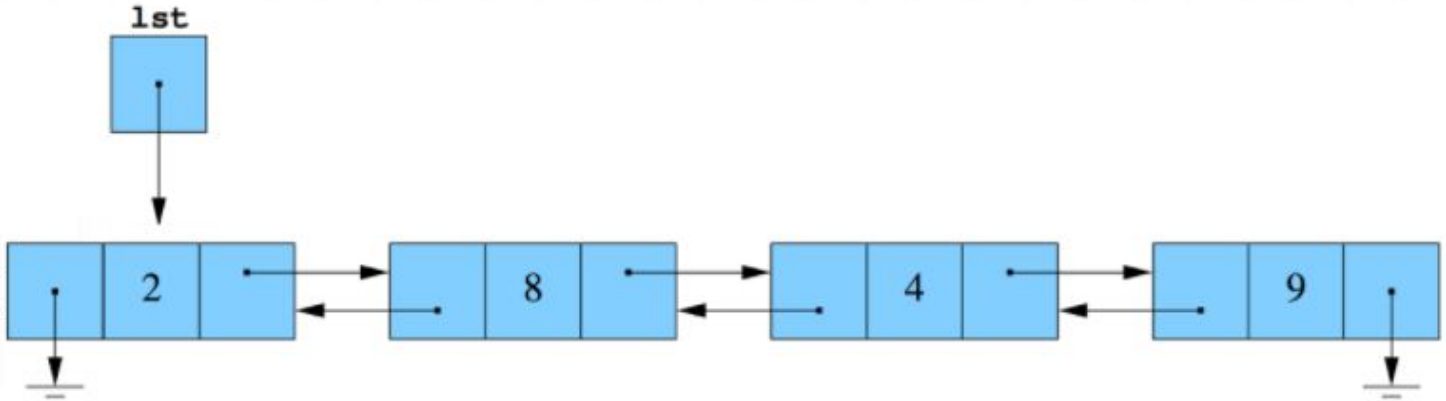
```
lst = lst->prox    // lst = p->prox;
if(lst != NULL)
    lst->ant = NULL;
```



Busca seguida de remoção







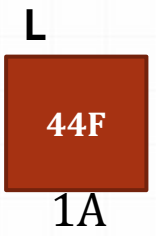
OPERAÇÃO: remoção de um elemento qualquer

Protótipo da função

*** parâmetros: valor a ser removido e ponteiro para a lista passado por referência.**

void busca_remove_dup (int, celula*&);

```
/*MAIN*/

int main()
{
    celula *L= NULL; //lista duplamente encadeada vazia
    int n;
…
    scanf("%d", &n);

    busca_remove_dup (   n ,   L);



…
}
```



Busca seguida de remoção

```
/* Recebe um número inteiro x e uma lista duplamente encadeada lst (por referência)
e remove da lista a primeira célula que contiver x, se tal célula existir */
void busca_remove_dup(int x, celula*& lst)
{
     celula *p;

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

    if(p == NULL)
        printf("\nNao encontrado");
    else{
        if(p->ant == NULL)   /*nó a remover é o primeiro*/
        {
            lst = lst->prox;
            if(lst != NULL)
                lst->ant = NULL;
        }
        else{
            p->ant->prox = p->prox;
            if(p->prox!=NULL)    /*caso p seja o último nó, seu prox é NULL*/
                p->prox->ant = p->ant;
        }
        free(p);
    }
}
```

lst (L)

lst é um apelido para a variável ‘L’

Busca seguida de remoção

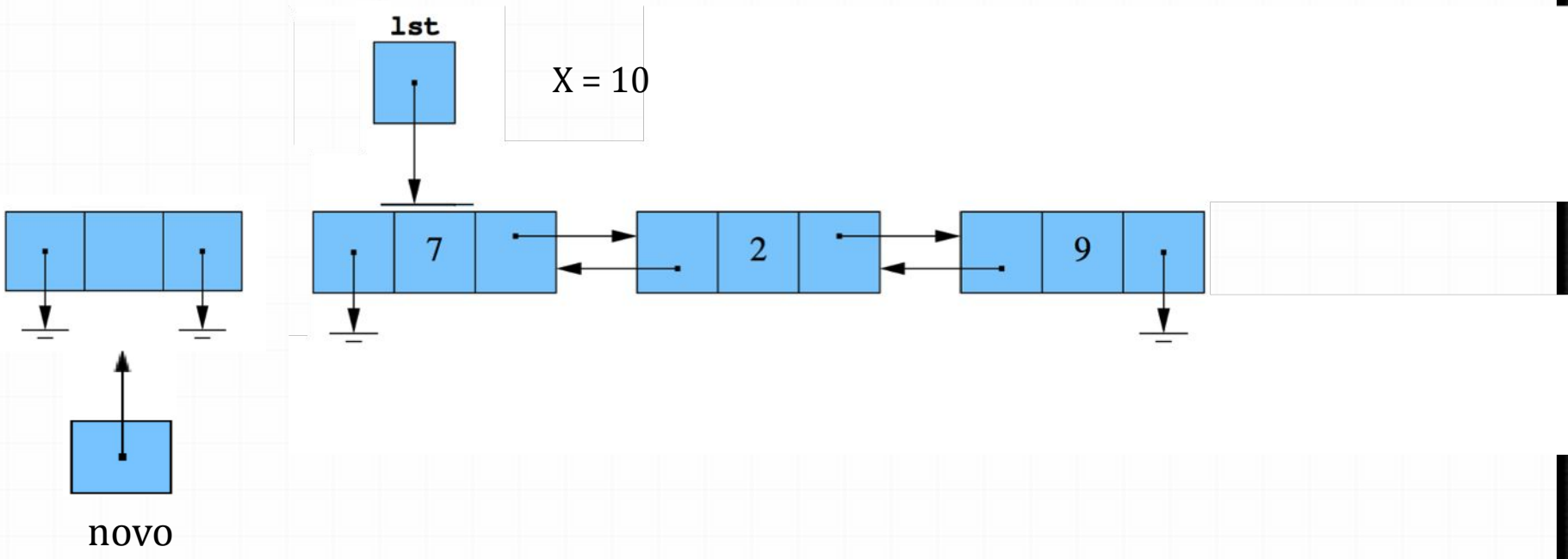
```
/* Recebe um número inteiro x e uma lista duplamente encadeada lst (por referência)
e remove da lista a primeira célula que contiver x, se tal célula existir */
void busca_remove_dup(int x, celula*& lst)
{
     celula *p;

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

    if(p == NULL)
        printf("\nNao encontrado");
    else{
        if(p->ant == NULL)   /*nó a remover é o primeiro*/
        {
            lst = lst->prox;
            if(lst != NULL)
                lst->ant = NULL;
        }
        else{
            p->ant->prox = p->prox;
            if(p->prox!=NULL)    /*caso p seja o último nó, seu prox é NULL*/
                p->prox->ant = p->ant;
        }
        free(p);
    }
}
```

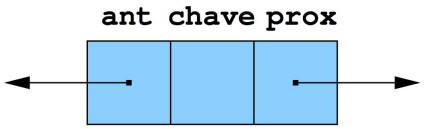
O(n)

Operação de inserção

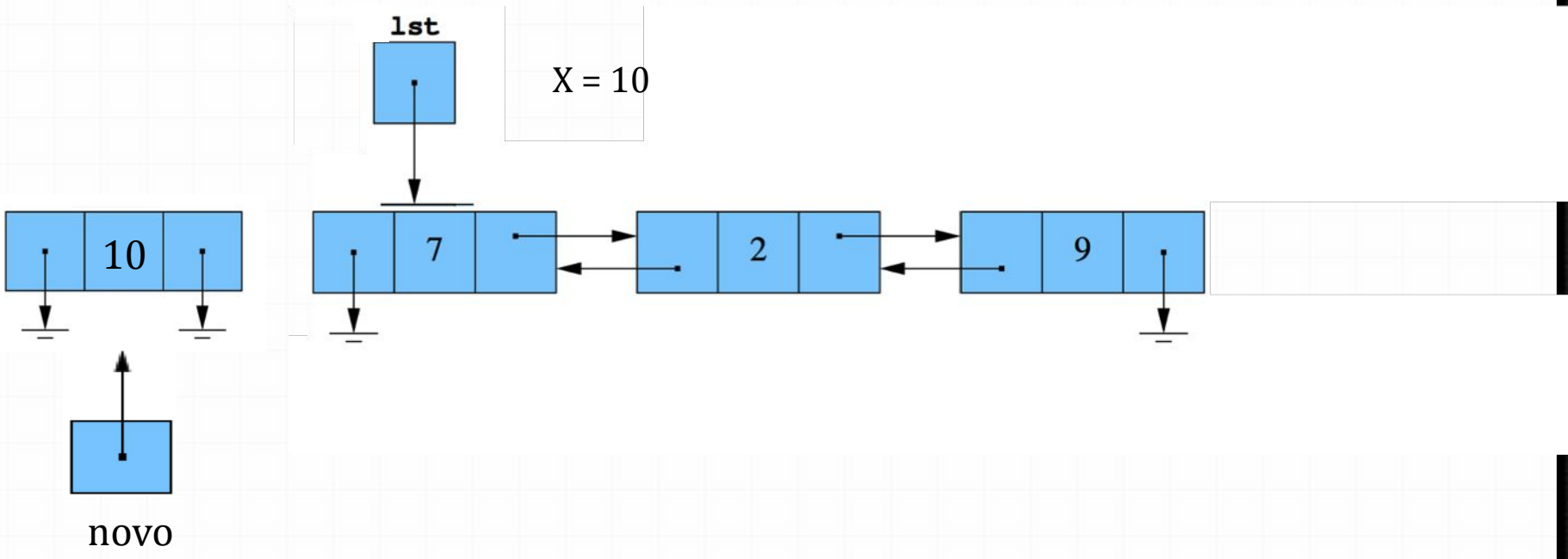
- **Inserção no início:**
  - **todo novo nó (célula) é inserido no início da lista**
- **Inserção no fim:**
  - **todo novo nó (célula) é inserido no fim da lista**

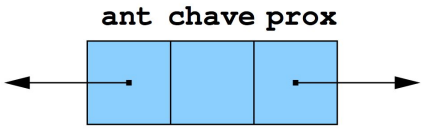
Inserção no início da lista



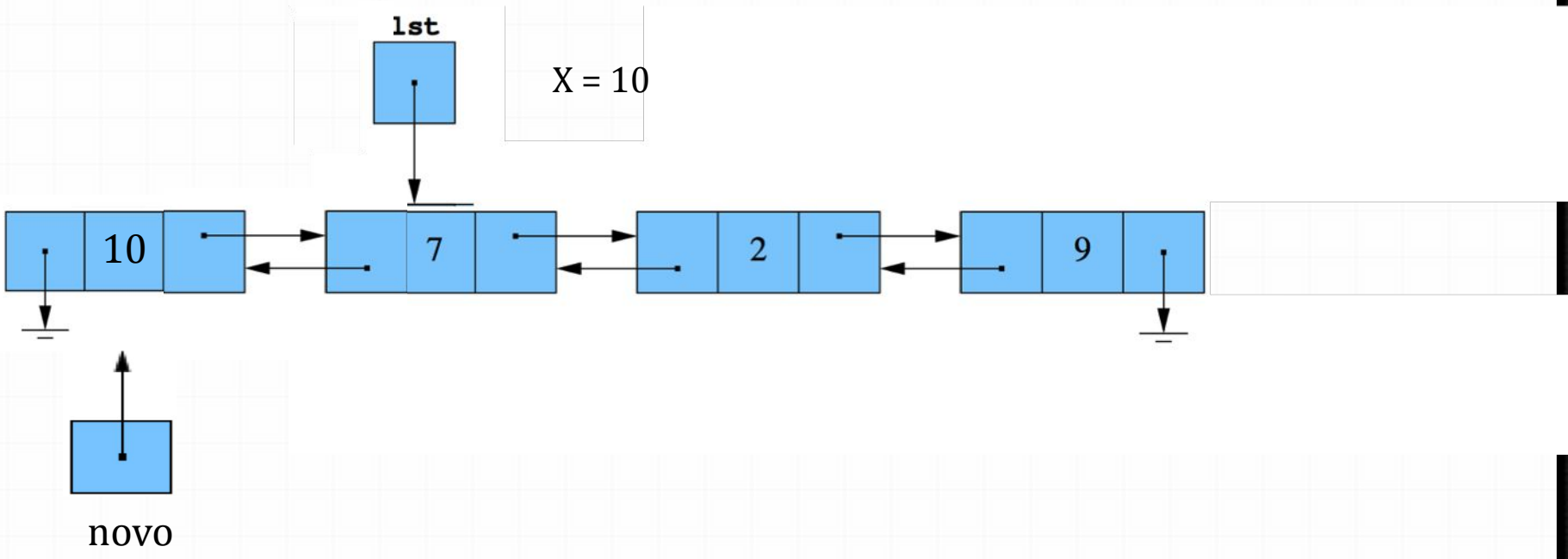


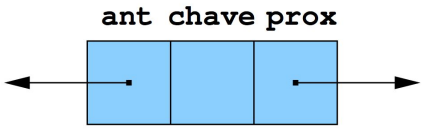
Inserção no início da lista



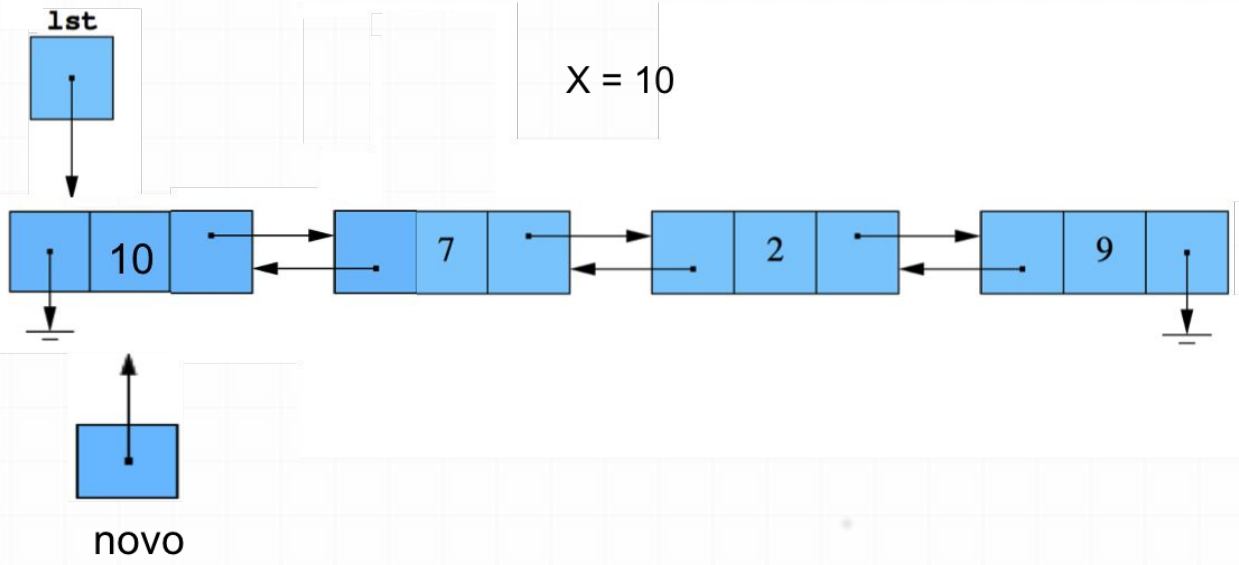


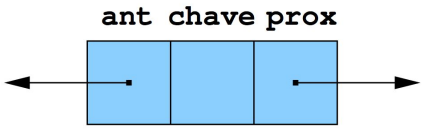
Inserção no início da lista



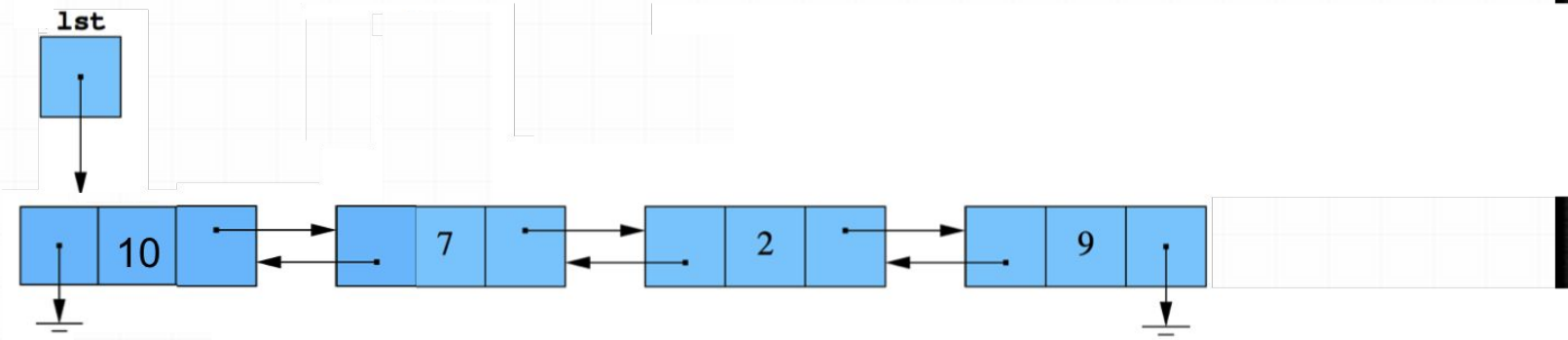


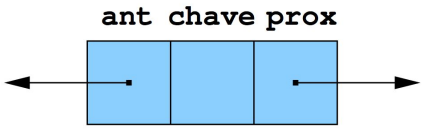
Inserção no início da lista





Inserção no início da lista





OPERAÇÃO: inserção no início

Protótipo da função

*** parâmetros: valor a ser inserido e ponteiro para a lista passado por referência.**

void inserir_inicio_dup(int, celula*&);

```
/*MAIN*/

int main()
{
    celula *L= NULL; //lista duplamente encadeada vazia
    int n;
…
    scanf(“%d”, &n);
    while( n!= 0 ){
        inserir_inicio_dup (   n ,   L);
        scanf(“%d”, &n);

    }
…
}
```

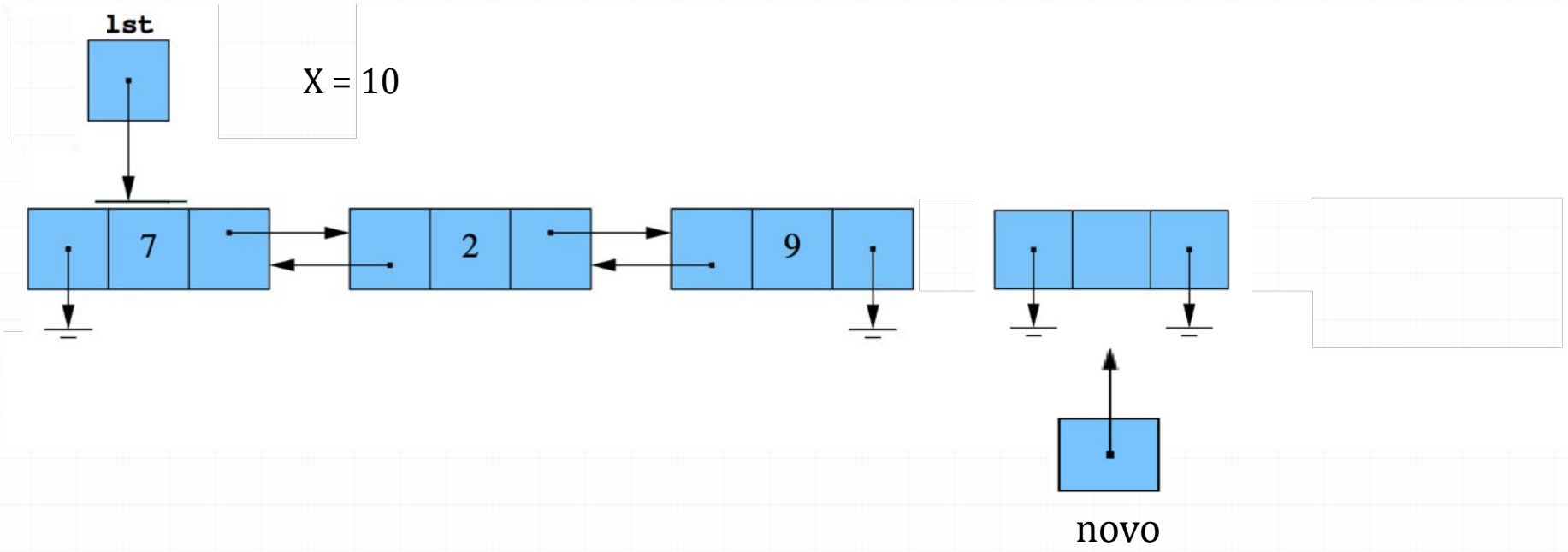
```
void inserir_inicio_dup (int x, celula*& lst)
{
    celula *novo;

    novo = (celula*) calloc(1, sizeof(celula));
    novo->chave = x;

     novo->prox = lst;
     if( lst != NULL)
         lst->ant = novo;

     lst = novo;
}
```

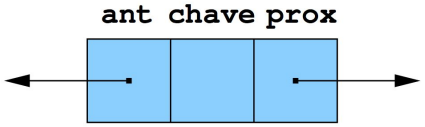
lst é um apelido para a variável 'L'

TEMPO DE EXECUÇÃO:

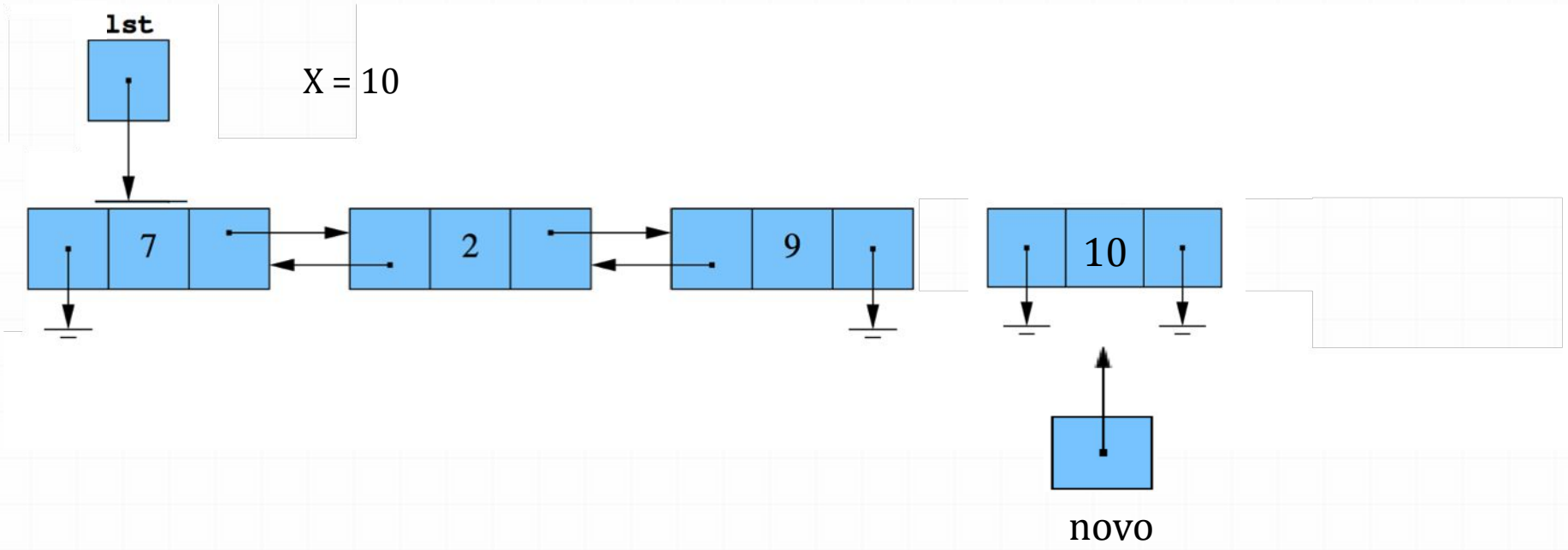
T(n) = O(1) – tempo constante

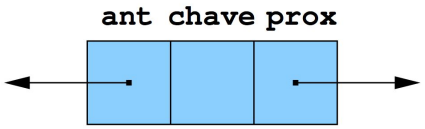
Inserção no fim da lista



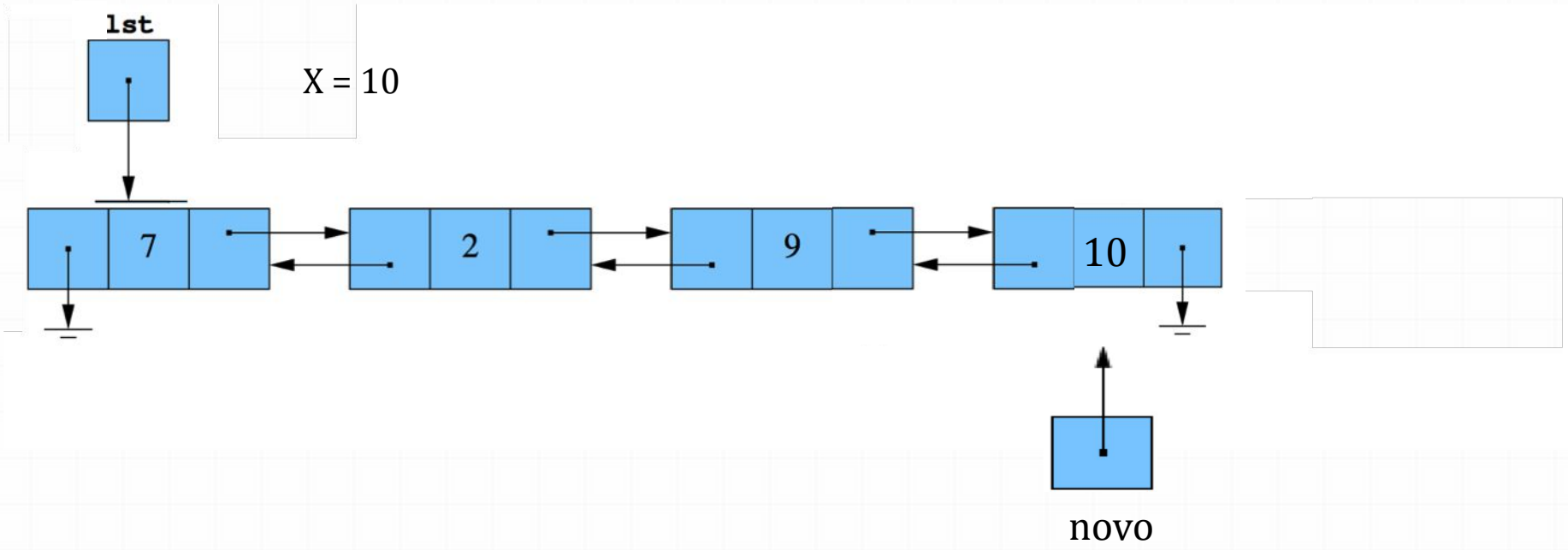


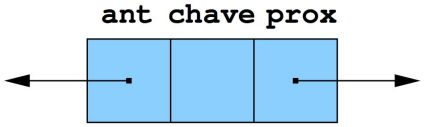
Inserção no fim da lista



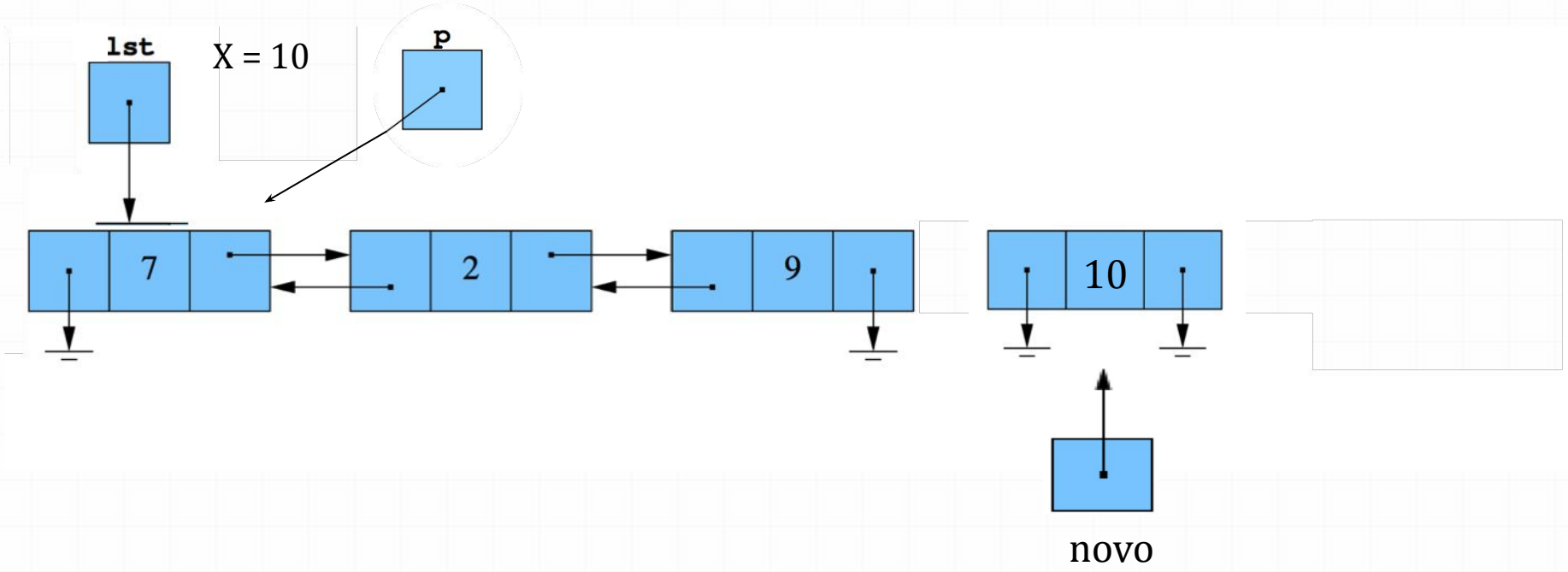


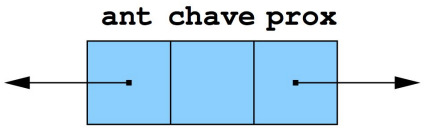
Inserção no fim da lista



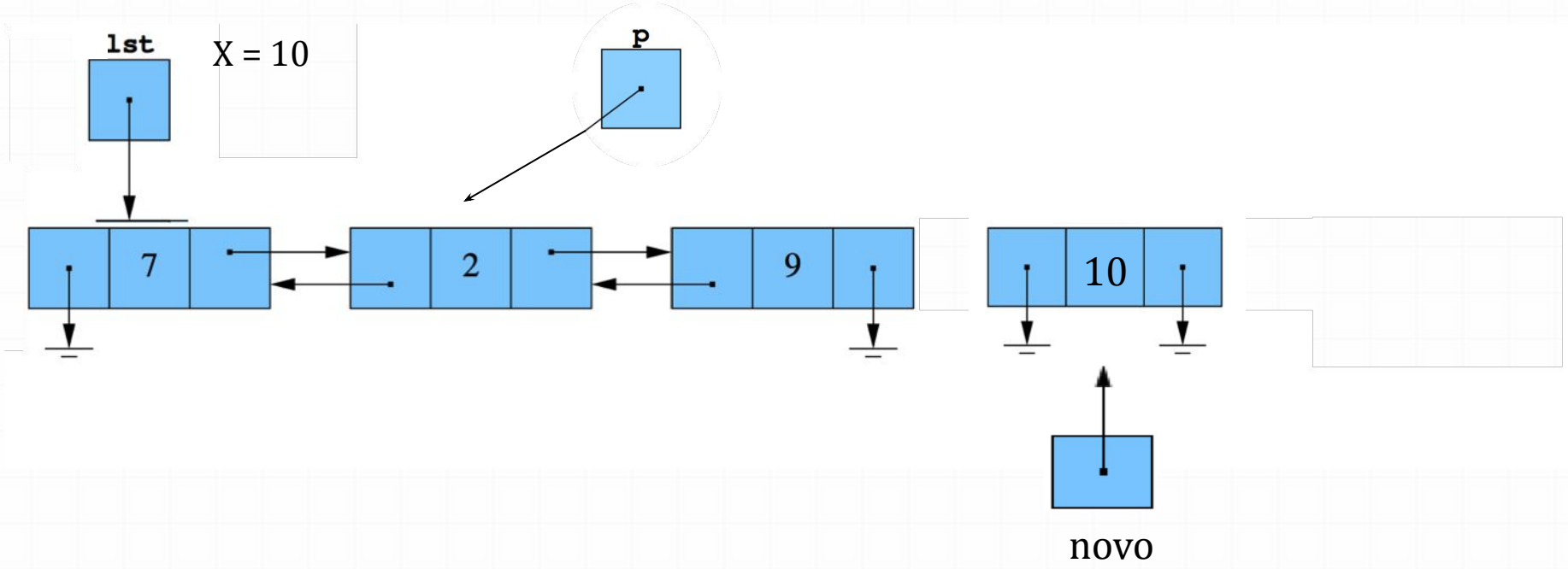


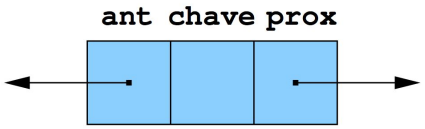
Inserção no fim da lista



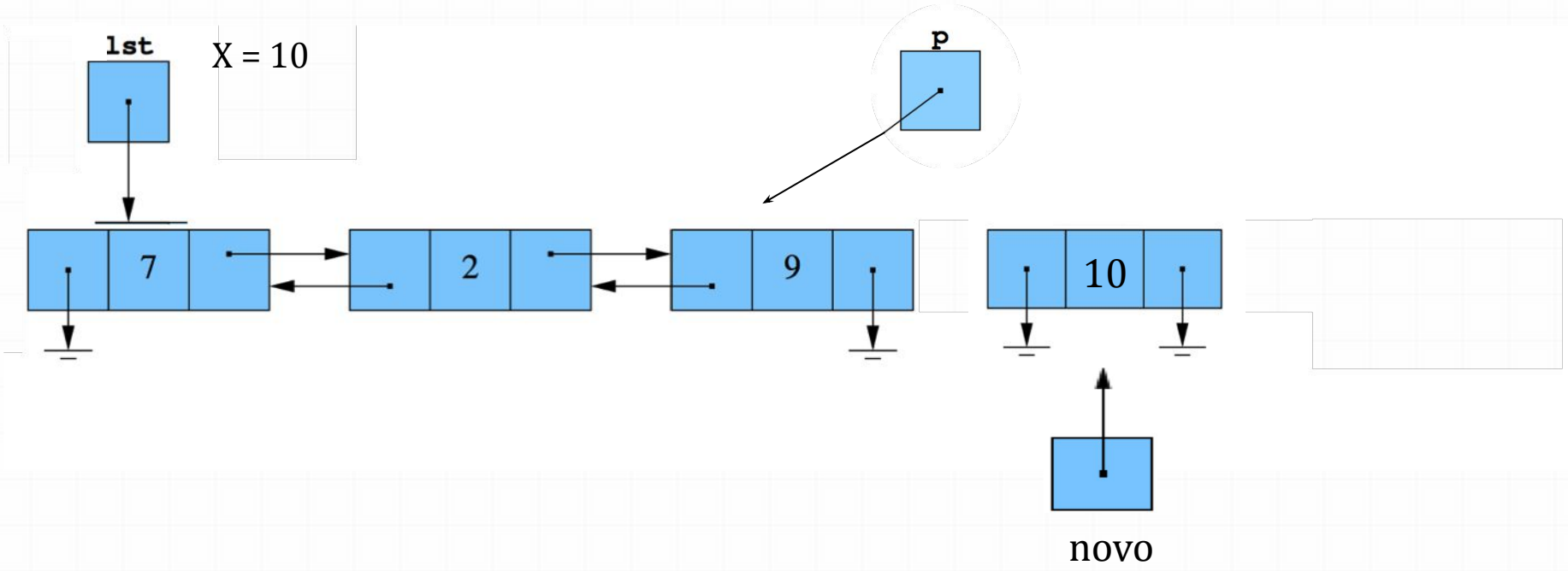


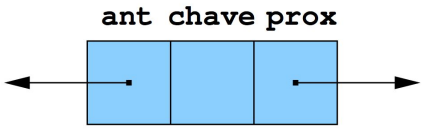
Inserção no fim da lista



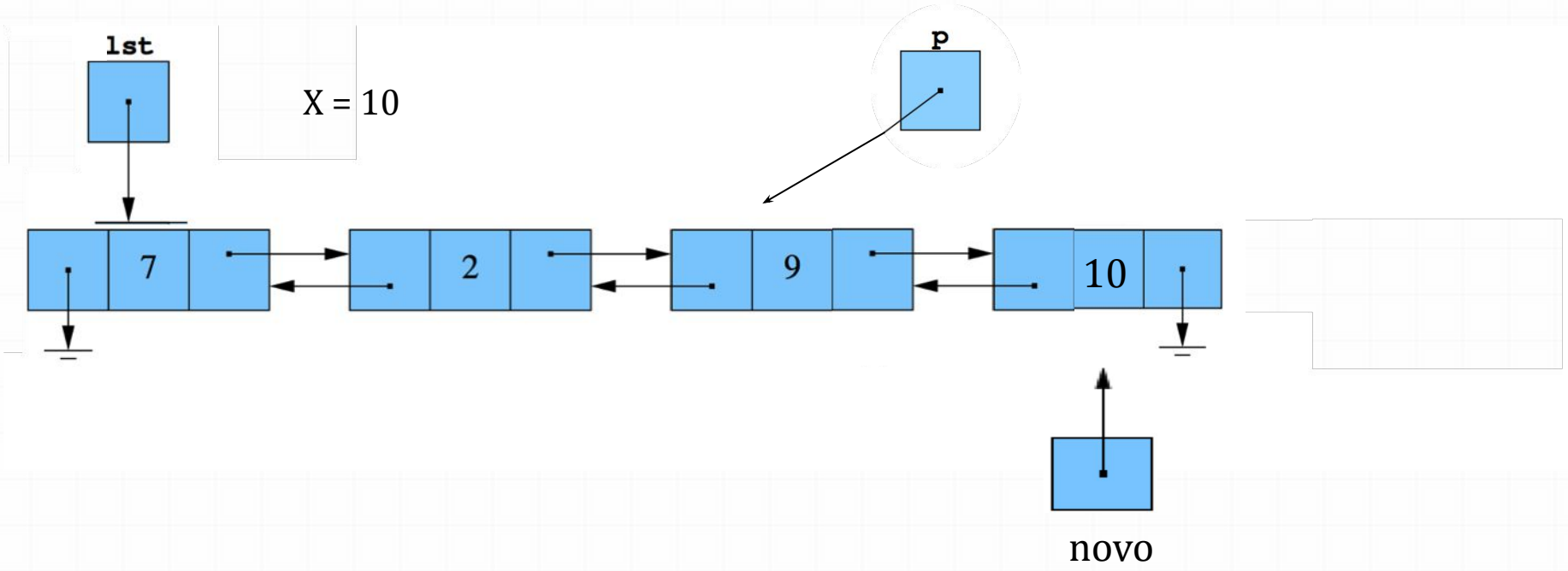


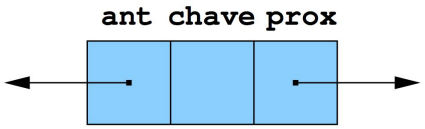
Inserção no fim da lista



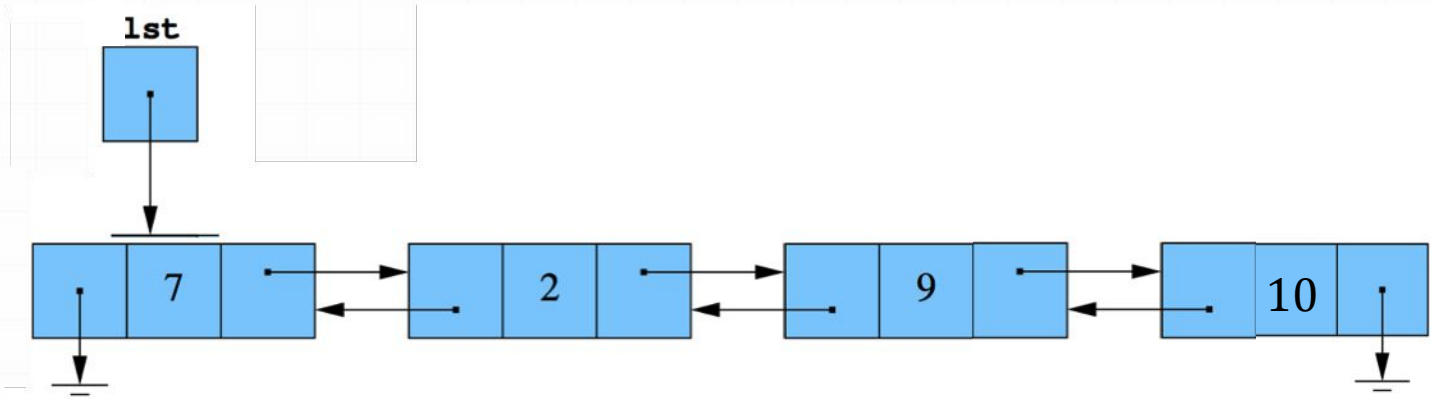


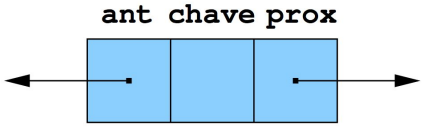
Inserção no fim da lista





Inserção no fim da lista





OPERAÇÃO: inserção no fim

Protótipo da função

*** parâmetros: valor a ser inserido e ponteiro para a lista passado por referência.**

void inserir_fim_dup(int, celula*&);

```
/*MAIN*/

int main()
{
    celula *L= NULL; //lista duplamente encadeada vazia
    int n;
…
    scanf(“%d”, &n);
    while( n!= 0 ){
        inserir_fim_dup (   n,   L);
        scanf(“%d”, &n);

    }
…
}
```

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

    novo = (celula*) calloc(1, sizeof(celula));
    novo->chave = x;

    if(lst == NULL)
        lst = novo;
    else{
        p = lst;
        while(p->prox != NULL)
            p = p->prox;

        p->prox = novo;
        novo->ant = p;
    }
}
```

lst é um apelido para a variável ‘L’

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

    novo = (celula*) calloc(1, sizeof(celula));
    novo->chave = x;

    if(lst == NULL)
        lst = novo;
    else{
        p = lst;
        while(p->prox != NULL)
            p = p->prox;

        p->prox = novo;
        novo->ant = p;
    }
}
```

lst (L)

O(n)

TEMPO DE EXECUÇÃO:

T(n) = O(n)