Unidade VII - Estruturas autorreferenciadas

Disciplina Linguagens de Programação I
Bacharelado em Ciência da Computação da Uerj
Professores Guilherme Mota e Gilson Costa

ANSI C

```c
#include <stdio.h>
int main ()
{
   printf("Hello World!");
   return 0;
}
```

Que assuntos serão abordados nesta unidade?

- Alocação dinâmica de structs
  - Arrays de structs
  - operador ->
- Listas encadeadas
  - Inserção
  - Busca
  - Exclusão
- Árvore Binária
  - Inserção sem balanceamento
  - Busca
  - Exclusão total

Alocação dinâmica de structs

Array estático de structs

```
struct coord{
   short x,y;
};

struct coord clist[2];



...

clist[1].x = 10;
clist[1].y = 5;
```

Array dinâmico de structs

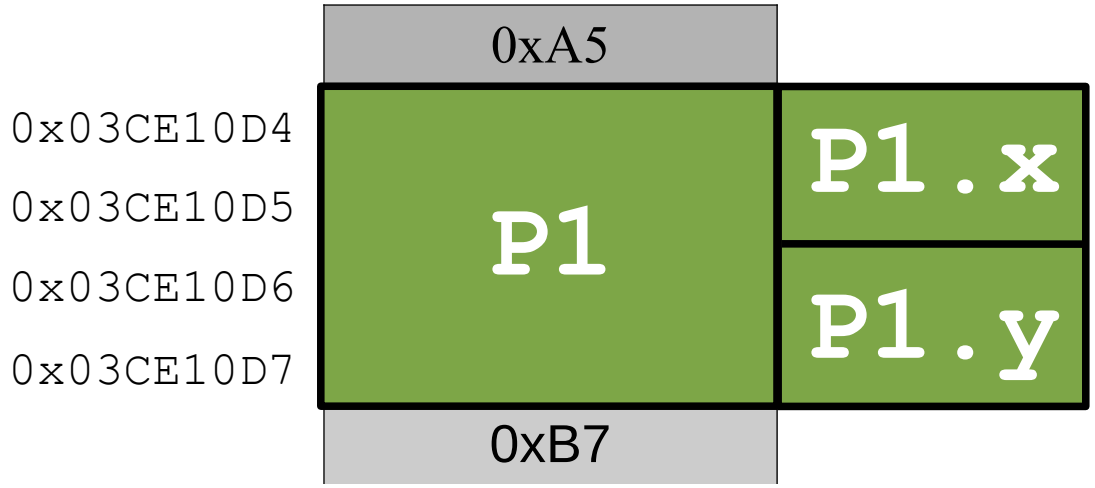
```
struct coord{
   short x,y;
};

struct coord * clist;
   ...
clist = malloc (2*sizeof(*clist));

for(int i=0; i<2; i++) {

   clist[i].x = i;
   (*(clist+i)).y = 2 * i;
   printf("X = (%d, %d)\n",(clist+i)->x,(clist+i)->y);
}
free(clist);
```

Operador . Vs Operador ->

- Em uma `struct` alocada de forma automática o operador `.` é usado para acessar os seus membros.

- Este operador apenas adiciona o deslocamento ao endereço base associado ao nome da `struct`.

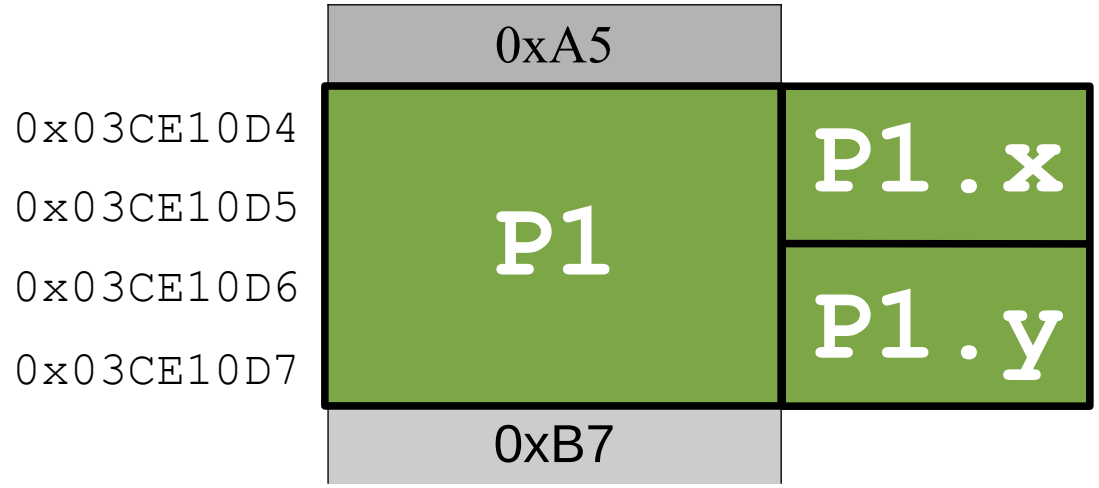
```
struct coord{
   short x,y;
};

struct coord P1;
P1.x = 10;
P1.y = 20;
```



Operador . Vs Operador ->

- Para referências através de um apontador por uma `struct` pode ser usado o operador `->`
- O operador `->` busca o endereço base da `struct` na variável apontador, aplica a indireção e depois aplica o deslocamento para acessar o campo selecionado.

```
struct coord{
    short x,y;
};

struct coord *pCoord, P1;
pCoord = &P1;
(*pCoord).x = 10;
pCoord->y = 20;
```



Operador . Vs Operador ->

- Com matrizes de `structs`, os operadores [] ou * são usados inicialmente para fazer o deslocamento e a indireção para o elemento antes de acessar o membro usando o operador .

```
struct coord{
   short x,y;
};

struct coord Vetor[5];

Vetor[3].y = 10;

(*(Vetor+3)).y = 10;

(Vetor+3)->y = 10;
```

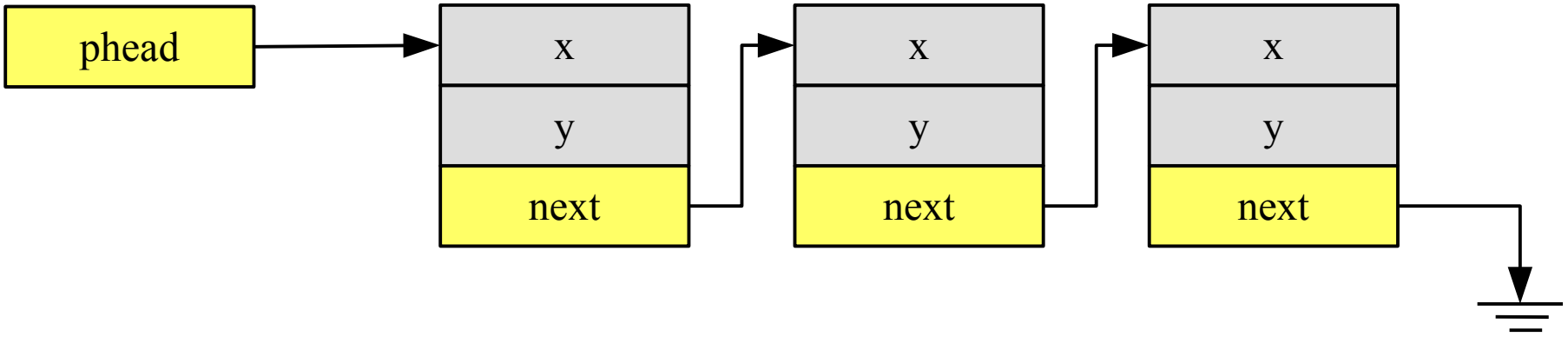
Listas Encadeadas

Vetores dinâmicos x Listas encadeadas

- Realloc é custoso para aumentar o tamanho de vetores alocados dinamicamente, pois envolve cópia de dados.
- No caso de exclusão de elementos, podemos marcar o elemento como eliminado, evitando a movimentação do vetor, mas o elemento apagado continuaria ocupando espaço
- No caso da inclusão em vetores dinâmicos, o realloc seria inevitável.
- Com listas encadeadas podemos adicionar e remover elementos sem precisar copiar os demais elementos para outros locais.
- Operações de ordenação são mais baratas em listas - os dados não são movimentados necessitando apenas ajustar os apontadores.
- Com listas não há elementos desnecessários na memória.

O que é uma lista encadeada?

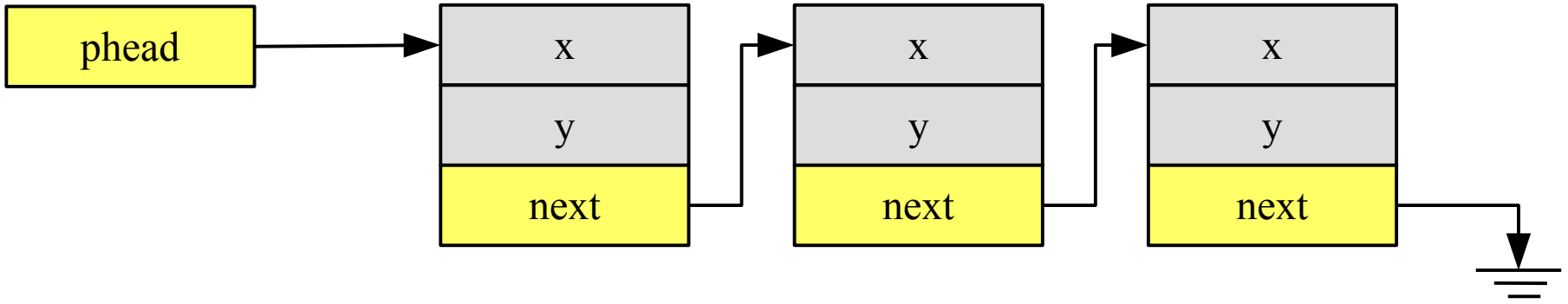
- Temos um apontador para o primeiro elemento da lista
- Cada elemento da lista aponta para o seguinte.



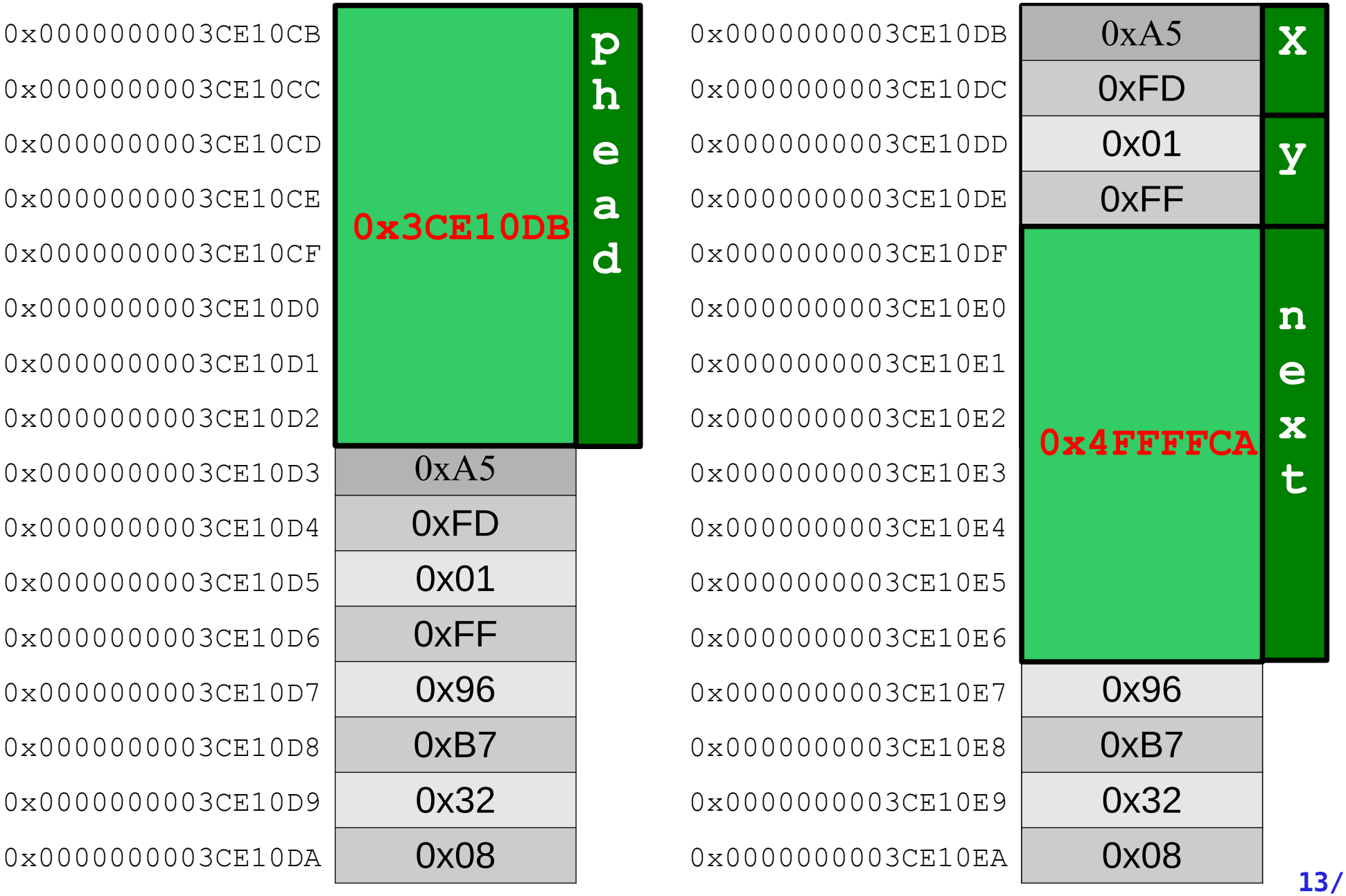

```
struct coord{
    short x;
    short y;
    struct coord * next;
};
```

Como construir listas encadeadas?

- Os elementos da lista são alocados dinamicamente na Heap
- A ordem física dos elementos na Heap é irrelevante.
- A lógica da ordenação dos elementos é dada pelo encadeamento explicitado pelos apontadores.
- Cada elemento da lista aponta para o seguinte.
- É necessário um apontador alocado automaticamente para manter o primeiro elemento da lista acessível.
- O último elemento da lista aponta para NULL.



Visão de uma lista encadeada na memória



Declaração de uma lista vazia

```
struct coord {
     short x,y;
     struct coord * next;
};

struct coord * phead = NULL;
```

Criando um elemento

```
struct coord * criaElem( short x, short y){

    struct coord * p = malloc(sizeof(* p));
    p->x = x;
    p->y = y;
    p->next = NULL;
    return p;
}
```

Inserção em Listas Encadeadas

Iteradores

- Apontadores são usados como iteradores.
- Iteradores servem para percorrer uma certa estrutura de dados.
- O tipo do iterador é apontador para o elemento.

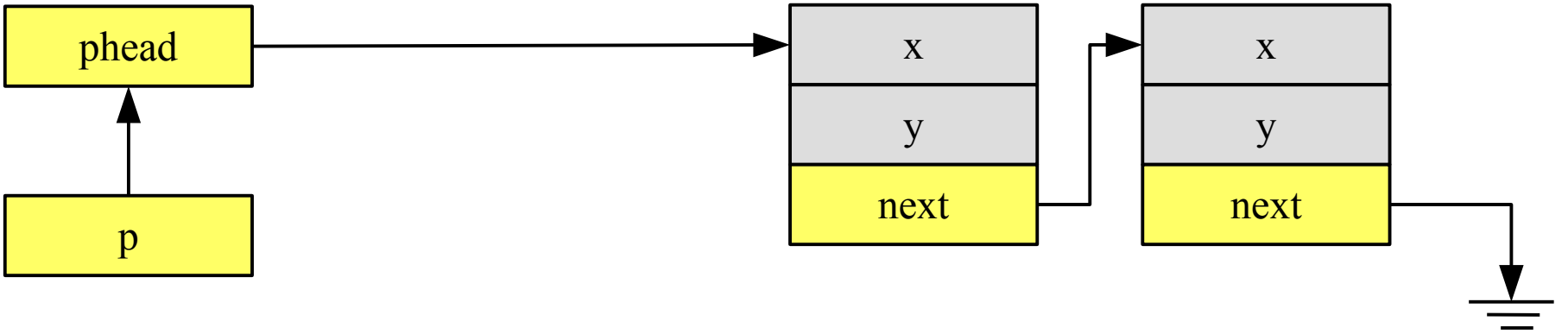
```
printf("%d",strlen("Guilherme"));


int strlen(char* s){

    char * p = s;

    while (*p != '\0');
       p++;

    return p-s;
}
```

Inserindo um elemento no inicio da lista

```
void insereInicio(struct coord ** p, short
x, short y) {

     struct coord * elem = criaElem(x, y);
     elem->next= *p;
     *p=elem;
}
```

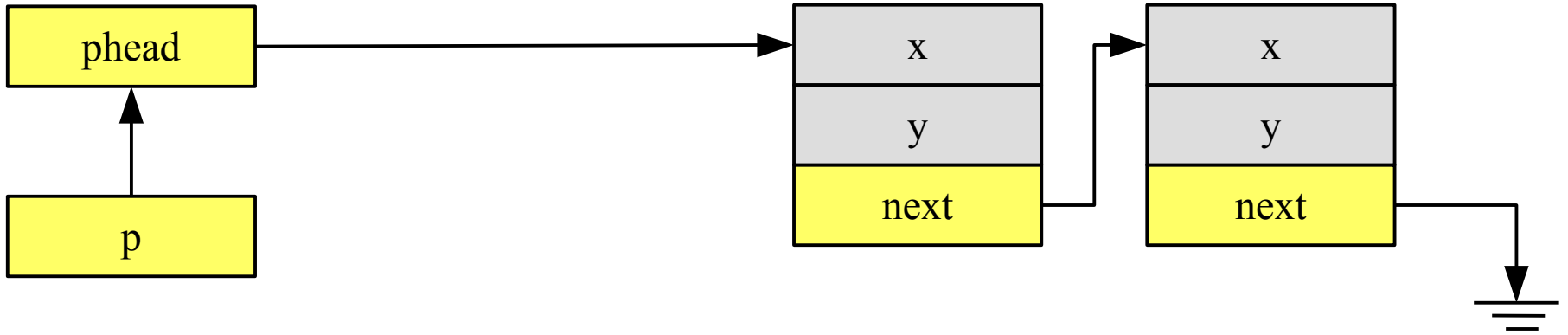
Inserindo um elemento no inicio da lista

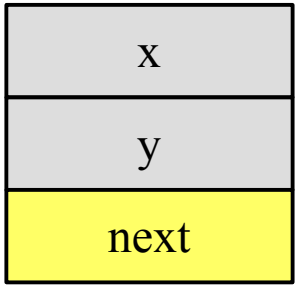
```
insereInicio(&phead, a,b);

  void insereInicio(struct coord ** p, short x, short y){

       struct coord * elem = criaElem(x,y);
       elem->next= *p;
       *p=elem;
  }
```



Inserindo um elemento no inicio da lista

```
insereInicio(&phead, a,b);

  void insereInicio(struct coord ** p, short x, short y) {

      struct coord * elem = criaElem(x,y);
      elem->next= *p;
      *p=elem;
  }
```

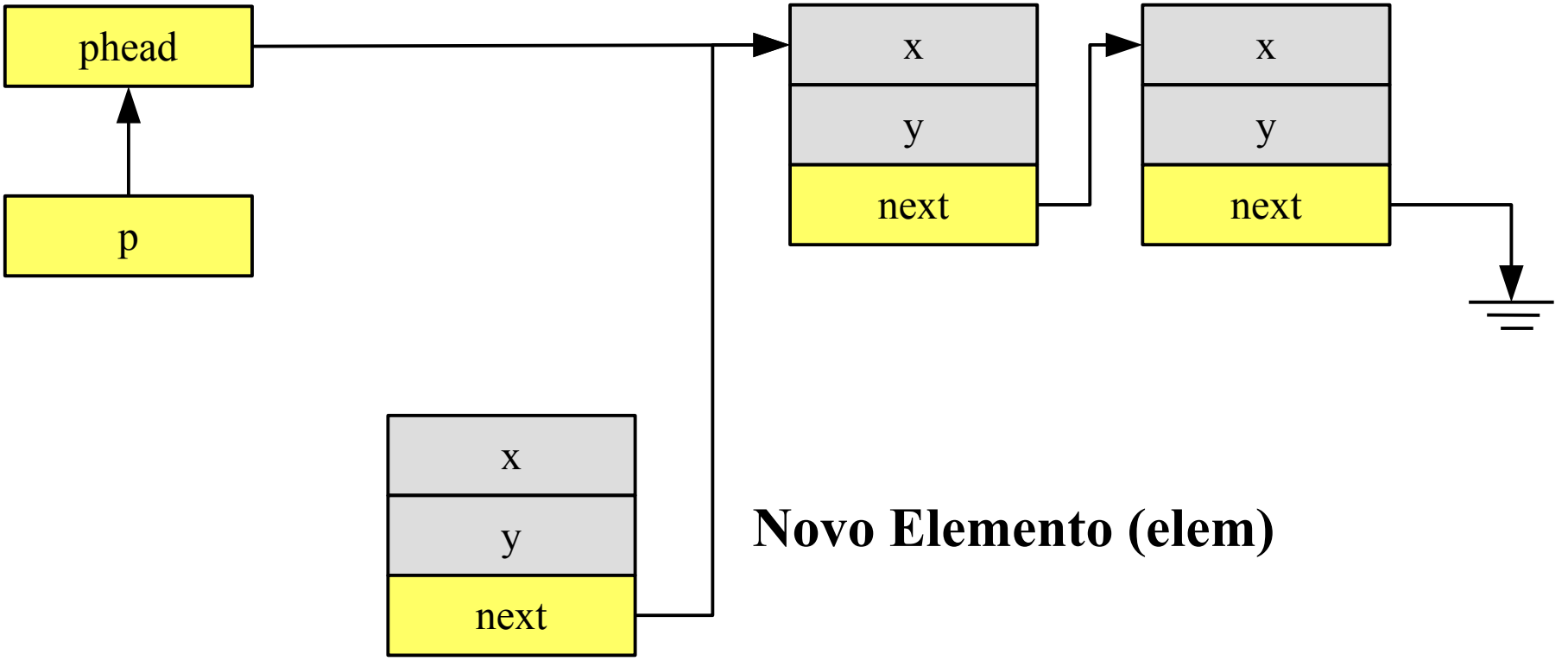




Novo Elemento (elem)

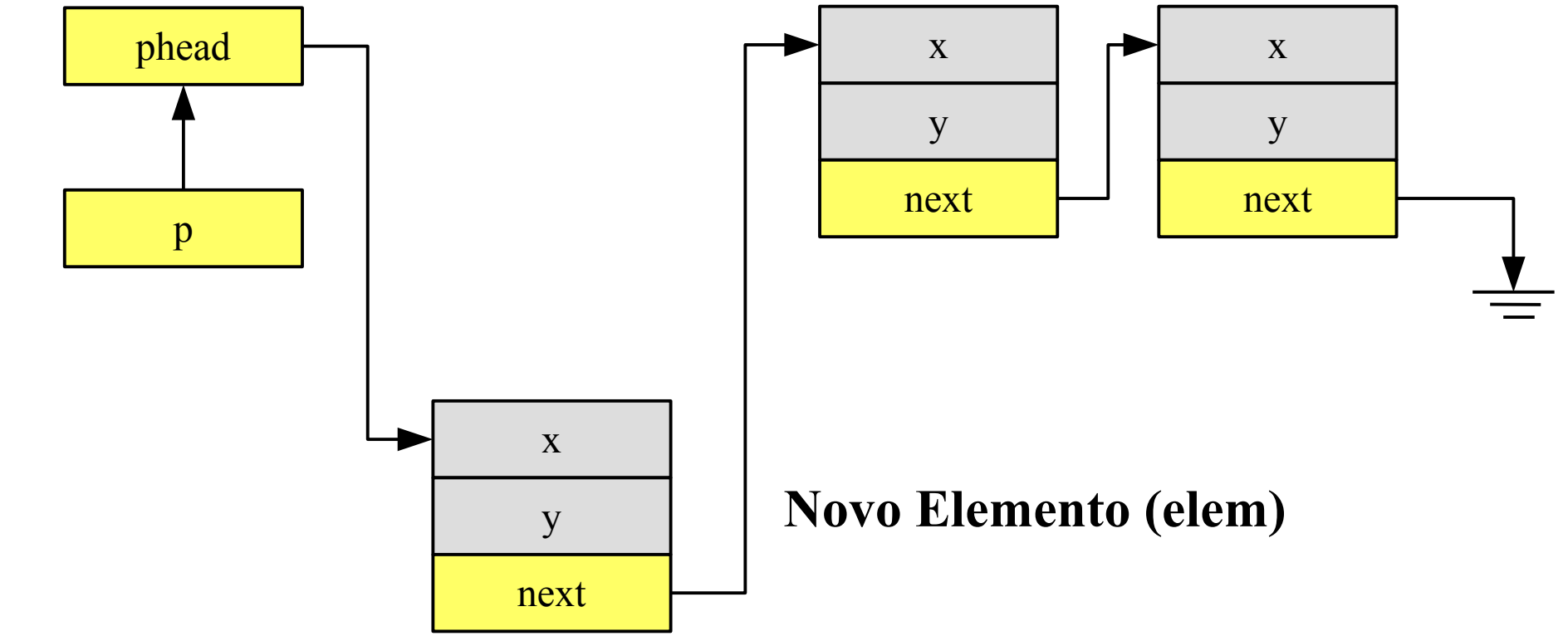
Inserindo um elemento no inicio da lista

```
insereInicio(&phead, a,b);

  void insereInicio(struct coord ** p, short x, short y) {

       struct coord * elem = criaElem(x,y);
       elem->next= *p;
       *p=elem;
  }
```



Inserindo um elemento no inicio da lista

```
insereInicio(&phead, a,b);

  void insereInicio(struct coord ** p, short x, short y) {

       struct coord * elem = criaElem(x,y);
       elem->next= *p;
       *p=elem;
  }
```



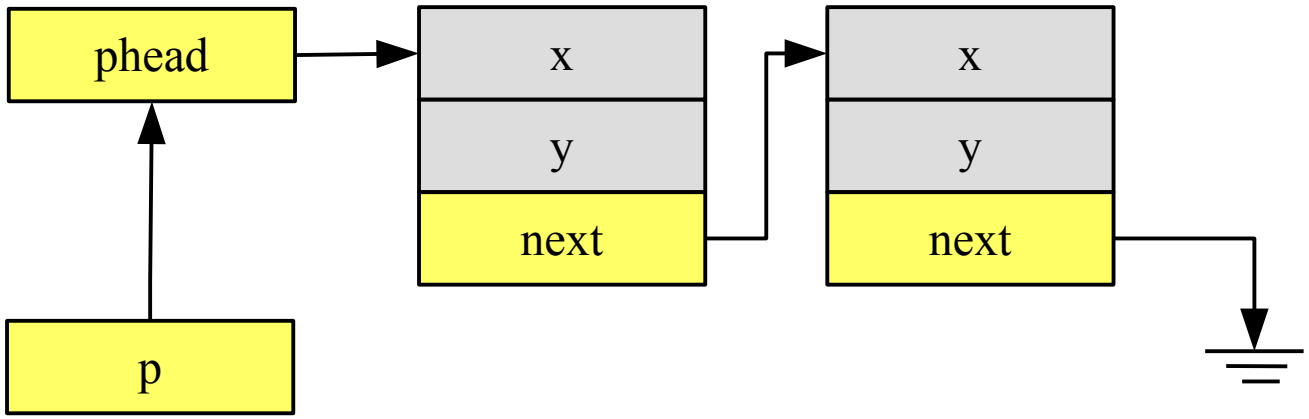
Inserindo um elemento no final da lista

```
void insereFim(struct coord ** p, short x,
short y) {

    if (*p)
        insereFim(&((*p)->next), x, y);
    else {

        struct coord * elem = criaElem(x,y);
        elem->next = NULL;
        *p=elem;
    }
}
```

Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

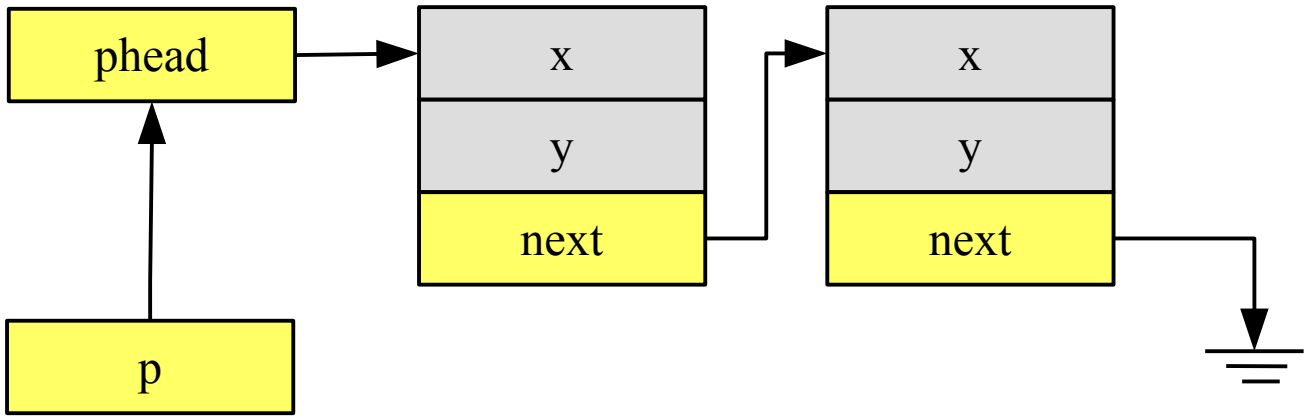
```
void insereFim(struct coord ** p, short x, short y){

    if (*p)
        insereFim(&((*p)->next), x, y);
    else {

        struct coord * elem = criaElem(x,y);
        elem->next= NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

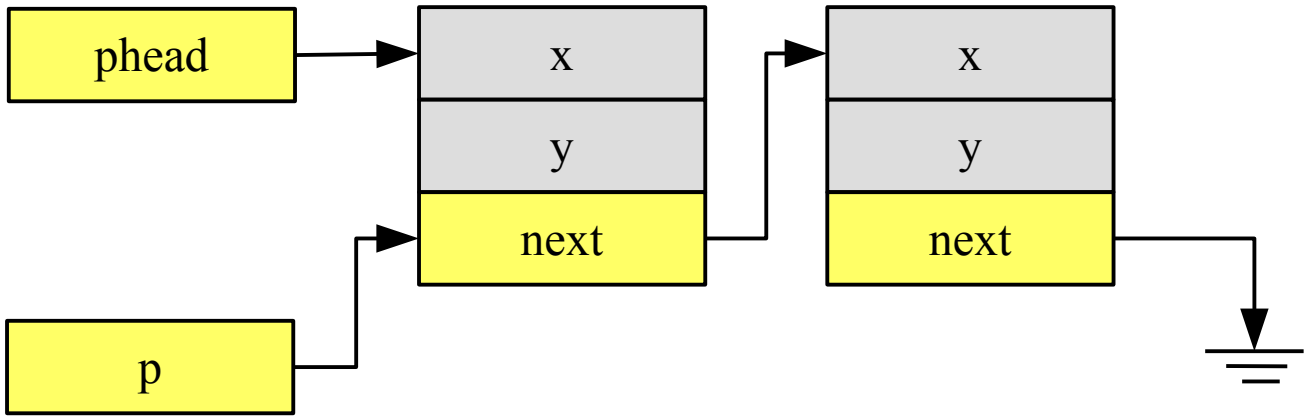
```
void insereFim(struct coord ** p, short x, short y){

    if (*p)
        insereFim(&((*p)->next), x, y);
    else {

        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

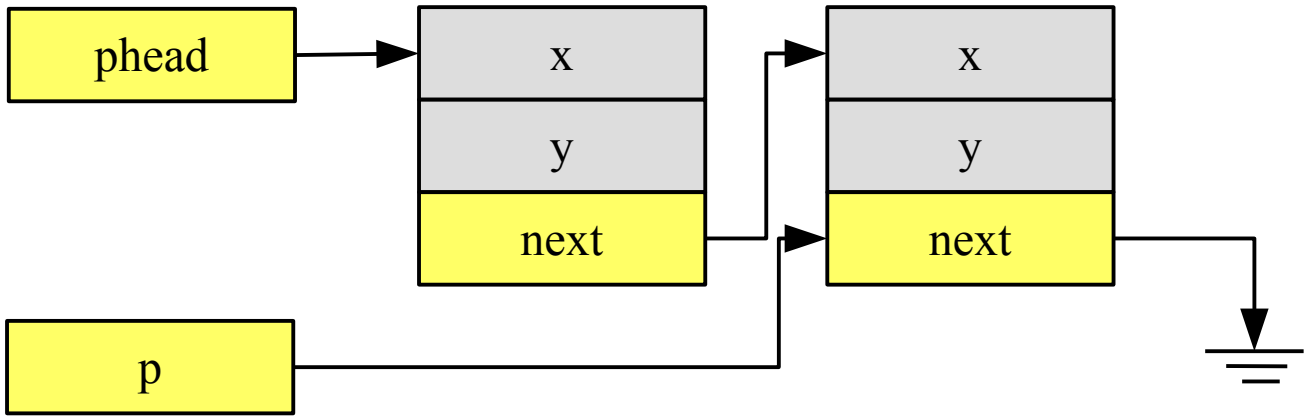
```
void insereFim(struct coord ** p, short x, short y){

    if (*p)
        insereFim(&((*p)->next), x, y);
    else {

        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

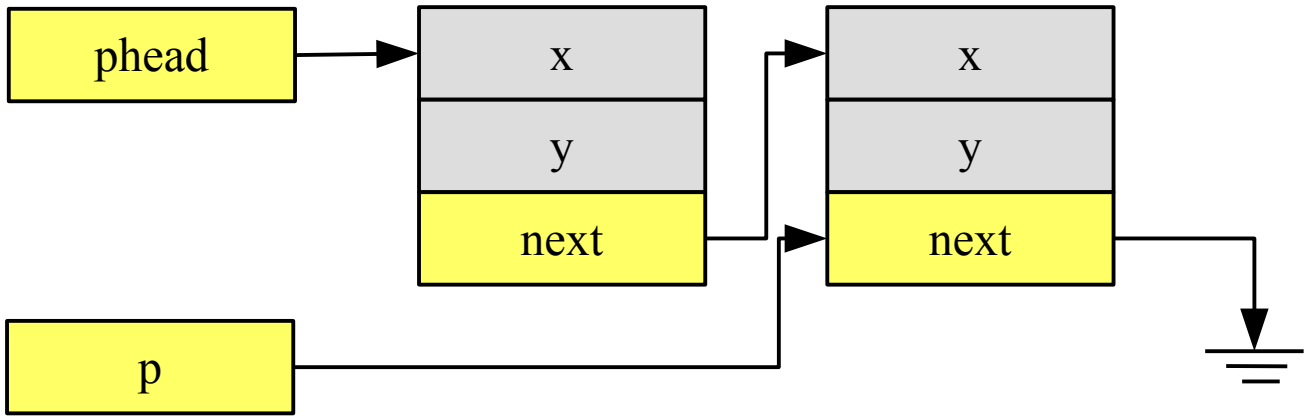
```
void insereFim(struct coord ** p, short x, short y) {

    if (*p)
        insereFim(&((*p)->next), x, y);
    else {

        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



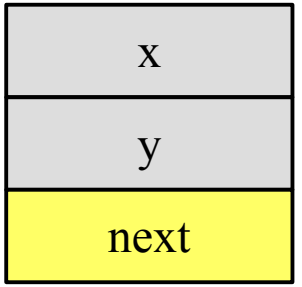
Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

```
void insereFim(struct coord ** p, short x, short y){

    if (*p)
        insereFim(&((*p)->next), x, y);
    else {

        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



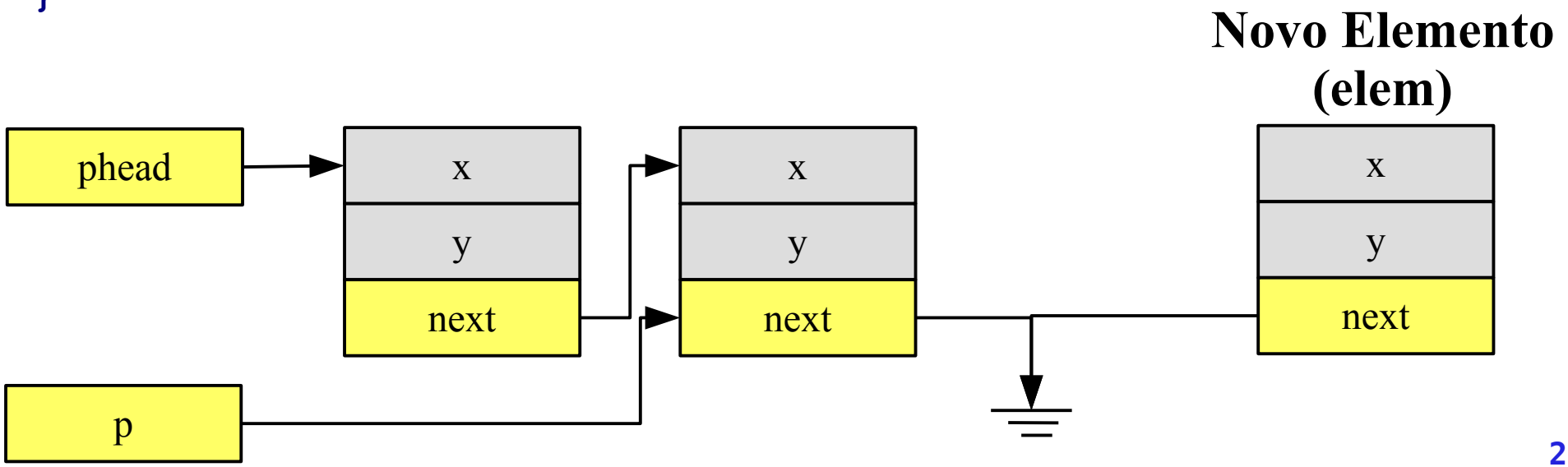
Novo Elemento (elem)



Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

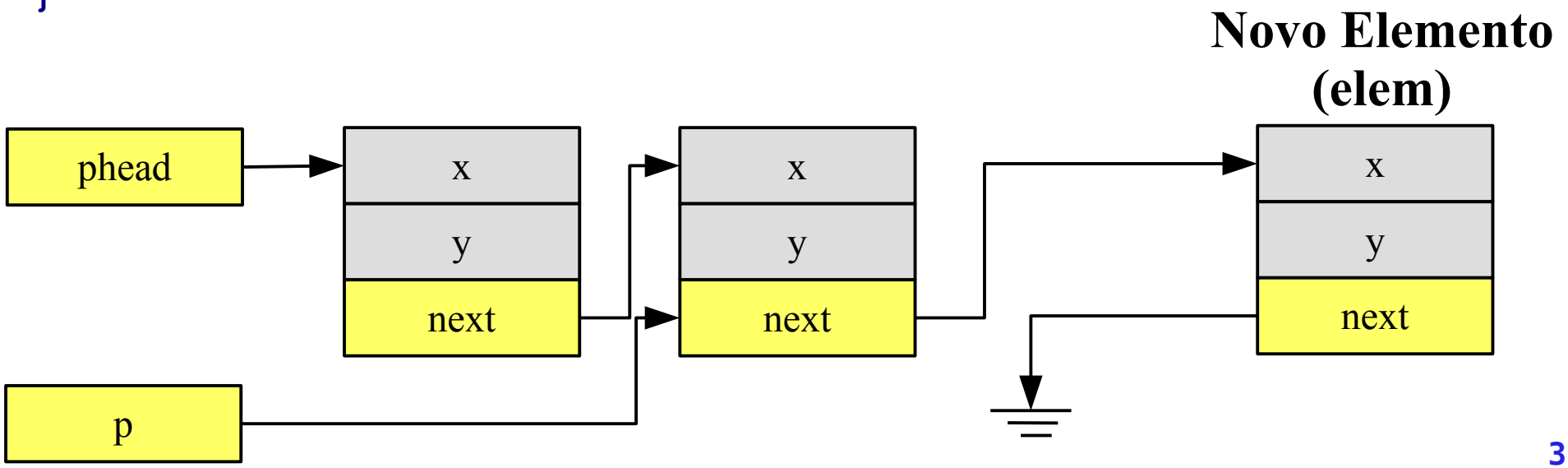
```
void insereFim(struct coord ** p){

    if (*p)
        insereFim(&((*p)->next));
    else {

        struct coord * elem = criaElem();
        elem->next = NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

```
insereFim(&phead, a,b);
```

```
void insereFim(struct coord ** p) {

    if (*p)
        insereFim(&((*p)->next));
    else {

        struct coord * elem = criaElem();
        elem->next = NULL;
        *p=elem;
    }
}
```



Percorrendo Listas Encadeadas

Imprimindo um elemento

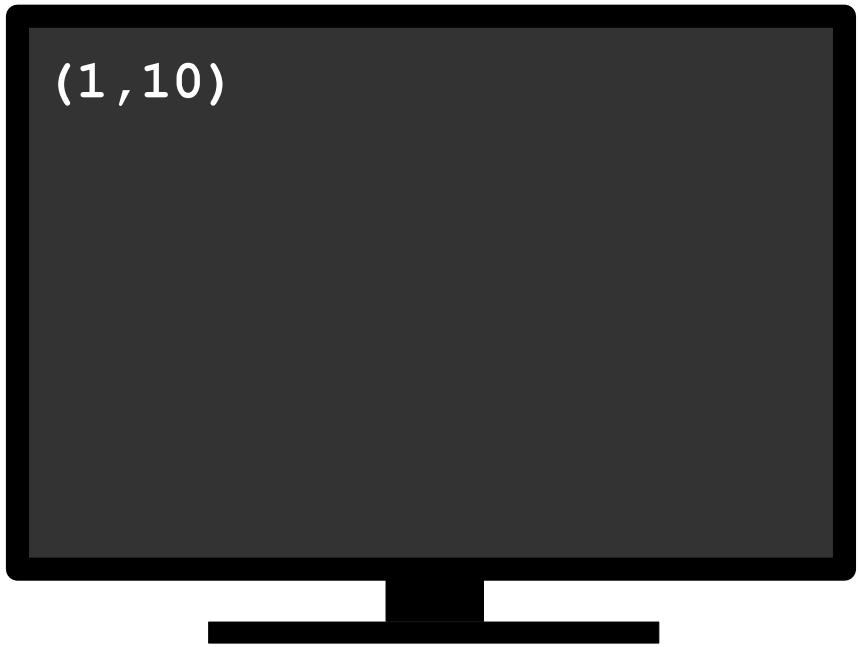
```
void imprimeElemento(struct coord * e) {

    if (e)
        printf("(%hd,%hd)\n", e->x, e->y);
    else
        printf("Elemento inexistente!\n");
}
```

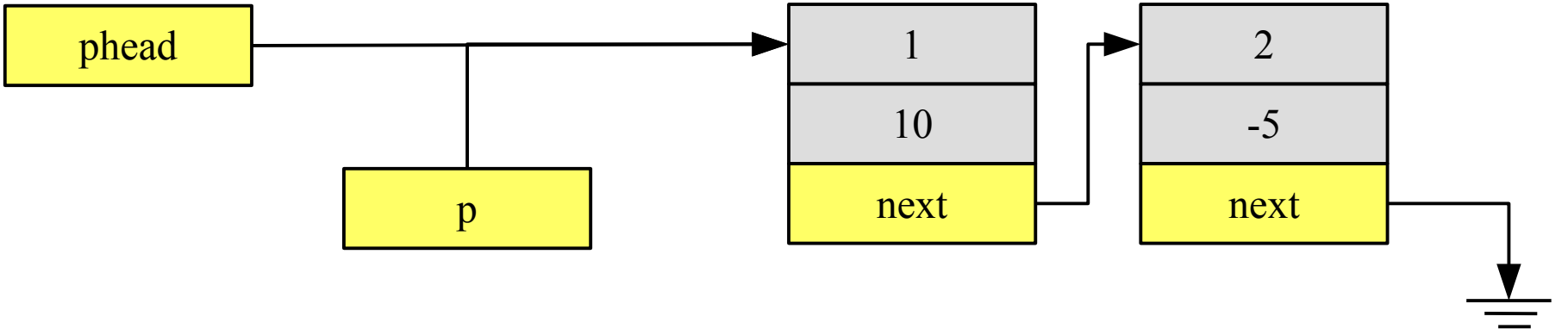
Imprimindo a lista

```
void imprimeLista(struct coord * p){

     if (p) {

          imprimeElemento(p);
          imprimeLista(p->next);
     }
     else
          printf("Fim da Lista\n");
}
```

Imprimindo a lista

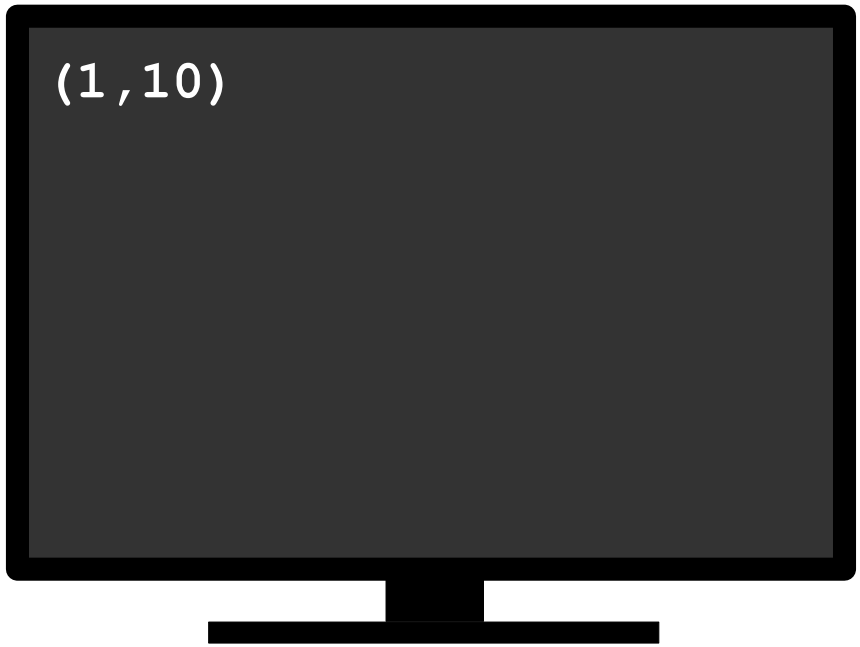
```
imprimeLista(phead);
   void imprimeLista(struct coord * p){

      if (p)   {

         imprimeElemento(p);
         imprimeLista(p->next);
      }
      else
         printf("Fim da Lista\n");
   }
```

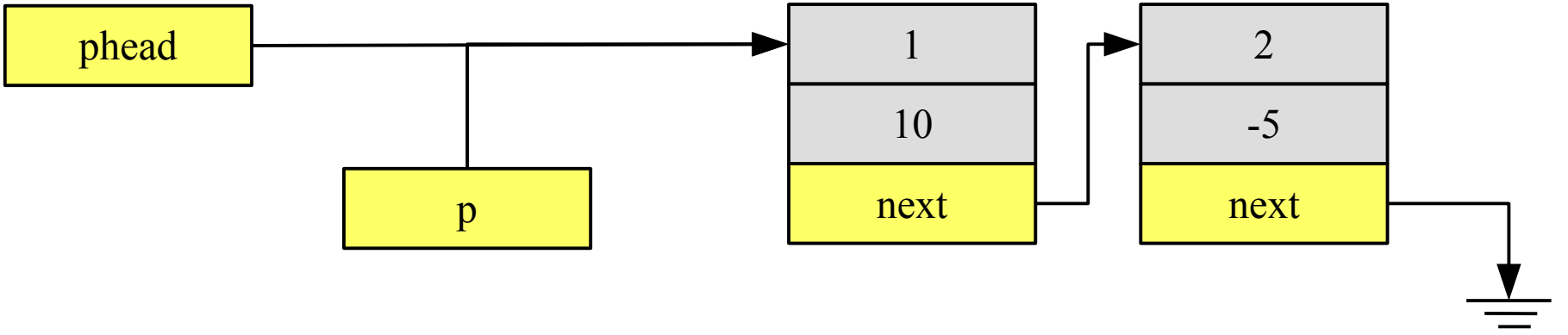




Imprimindo a lista

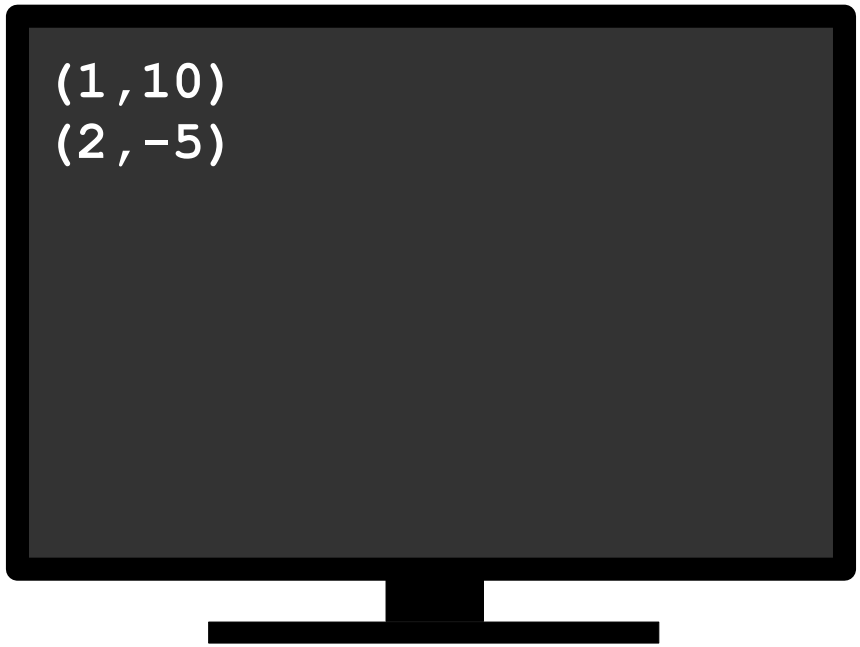
```
imprimeLista(phead);
   void imprimeLista(struct coord * p){

      if (p)   {

         imprimeElemento(p);
         imprimeLista(p->next);
      }
      else
         printf("Fim da Lista\n");
   }
```

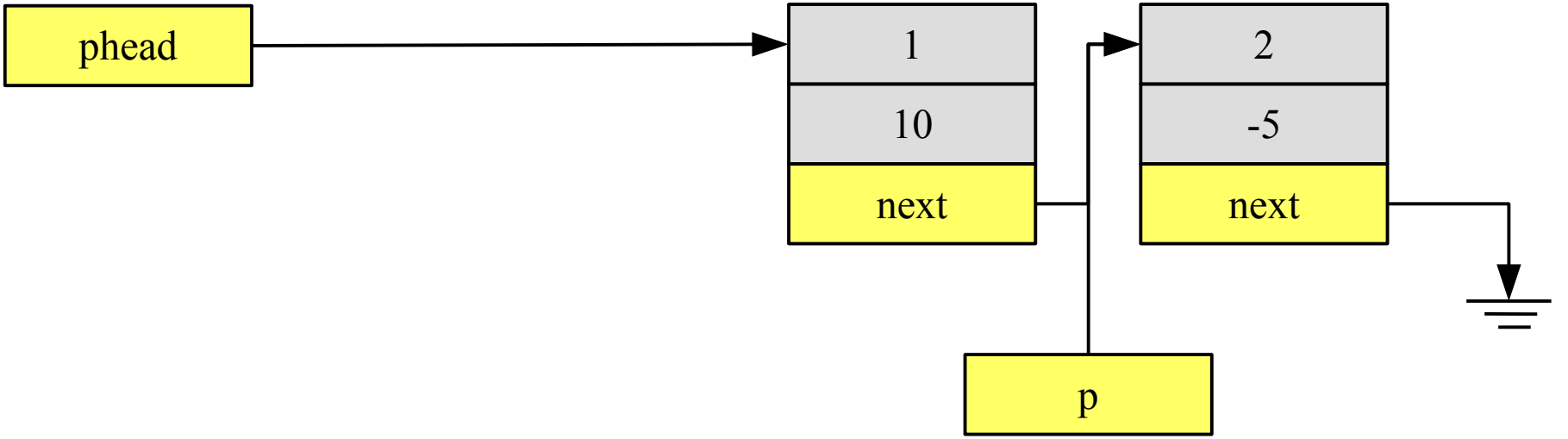




Imprimindo a lista

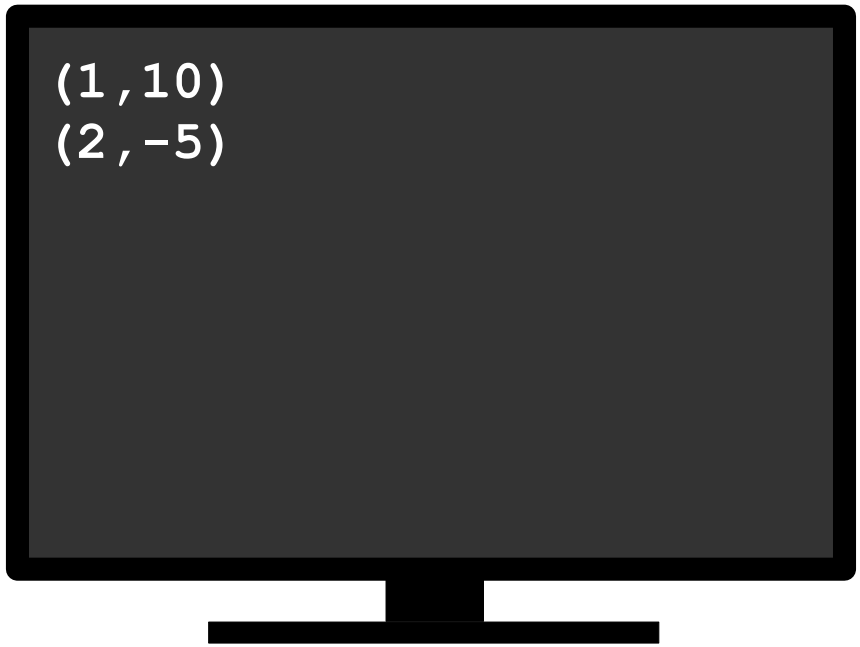
```
imprimeLista(phead);
   void imprimeLista(struct coord * p) {

      if (p) {

         imprimeElemento(p);
         imprimeLista(p->next);
      }
      else
         printf("Fim da Lista\n");
   }
```

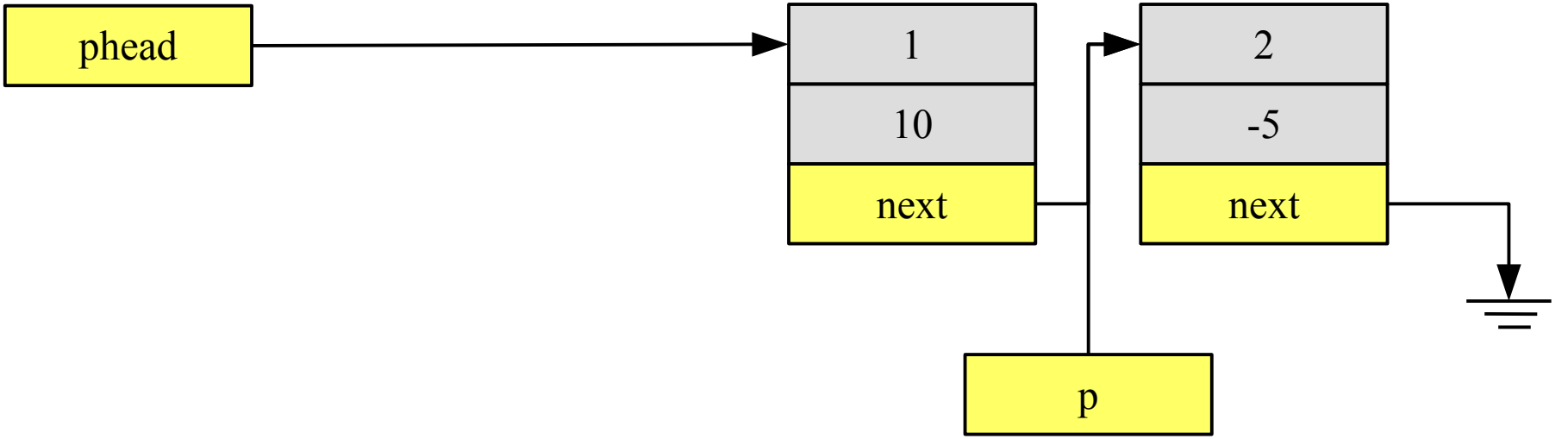




Imprimindo a lista

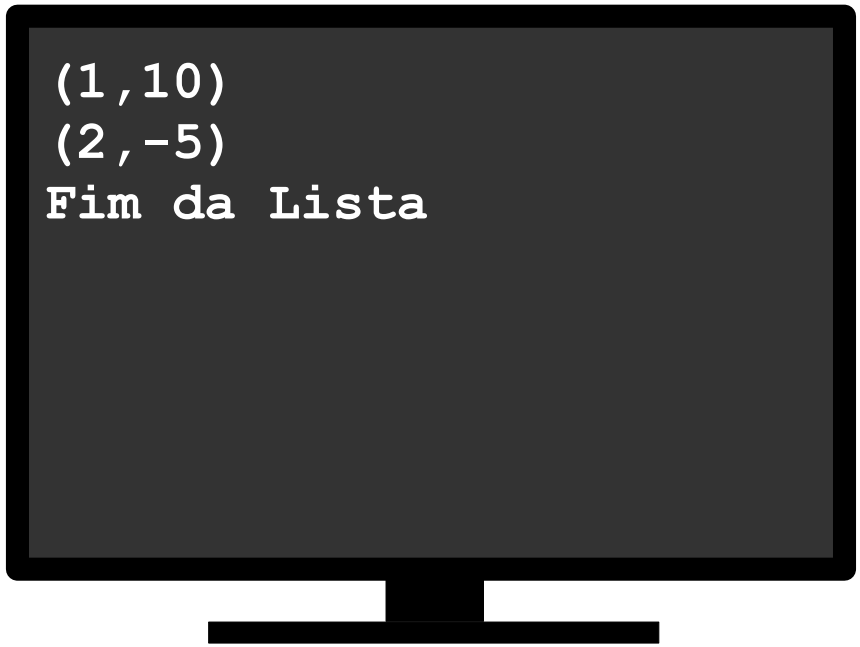
```
imprimeLista(phead);
   void imprimeLista(struct coord * p) {

      if (p) {

         imprimeElemento(p);
         imprimeLista(p->next);
      }
      else
         printf("Fim da Lista\n");
   }
```

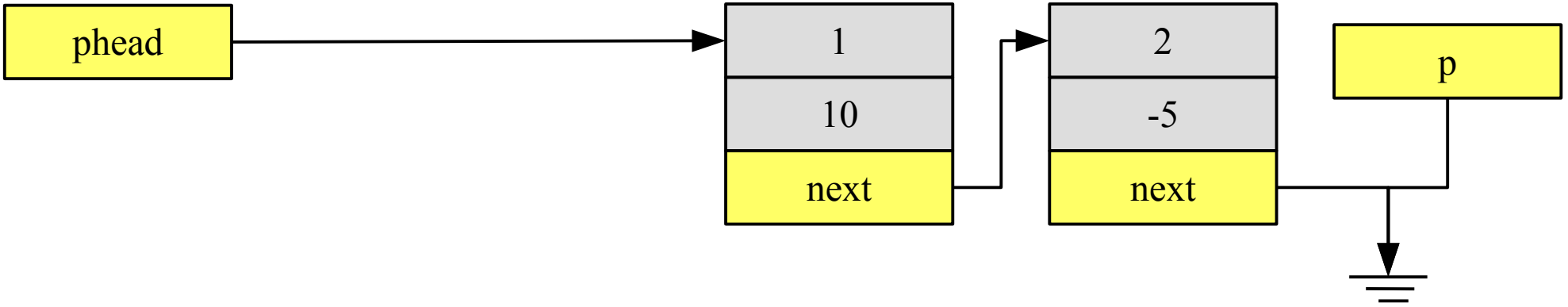




Imprimindo a lista

```
imprimeLista(phead);
   void imprimeLista(struct coord * p){

      if (p)   {

         imprimeElemento(p);
         imprimeLista(p->next);
      }
      else
      printf("Fim da Lista\n");
   }
```



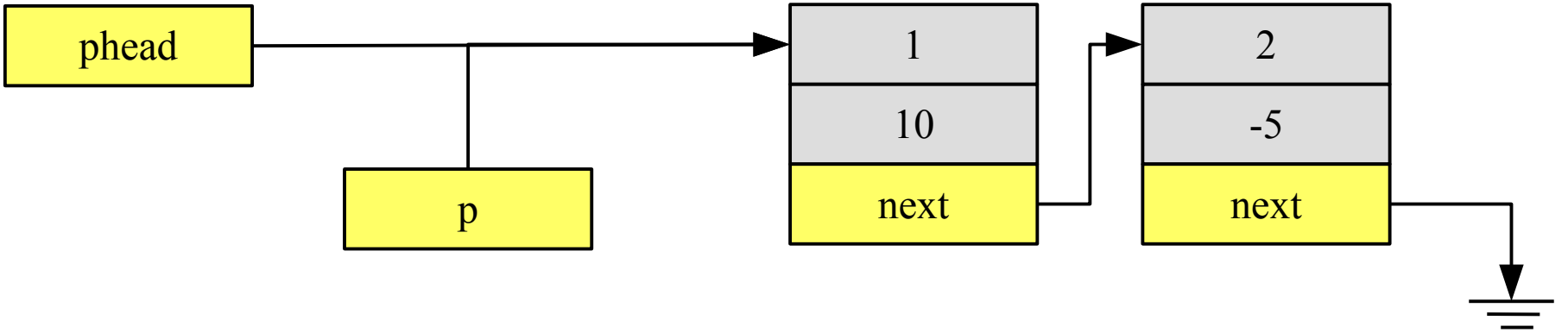


Buscando Elemento

```
struct coord * buscaCoord(int x, int y,
struct coord * p) {

    if (p)
        if ((x==p->x) && (y==p->y))
            return p;
        else
            return buscaCoord(x, y, p->next);
    else
        return NULL;
}
```

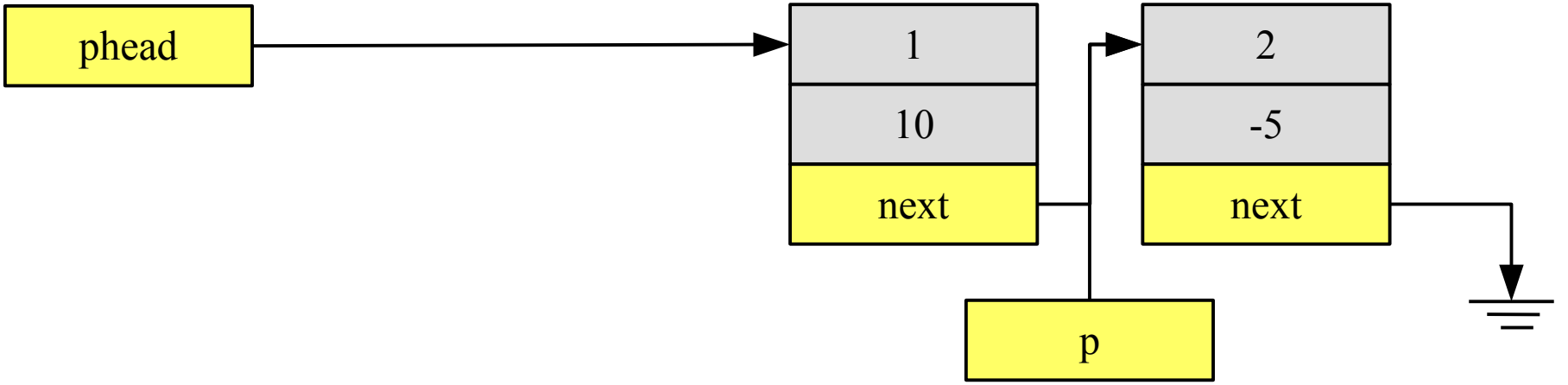
Buscando Elemento

```
imprimeElemento(buscaCoord(2,-5,phead));
  struct coord * buscaCoord(int x, int y, struct coord * p)
  {
     if (p)
        if ((x==p->x) && (y==p->y))
           return p;
        else
        return buscaCoord(x, y, p->next);
     else
        return NULL;
  }
```



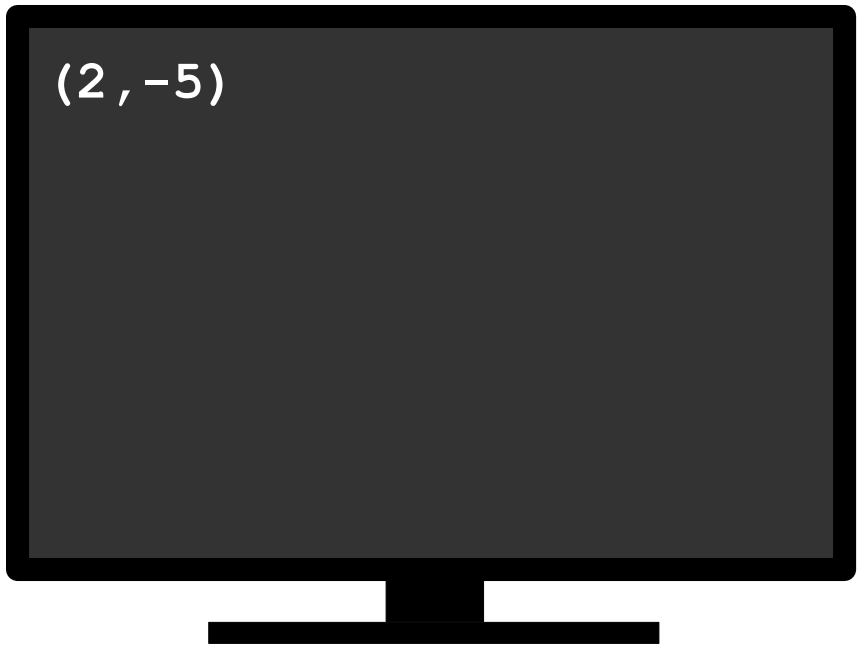
Buscando Elemento

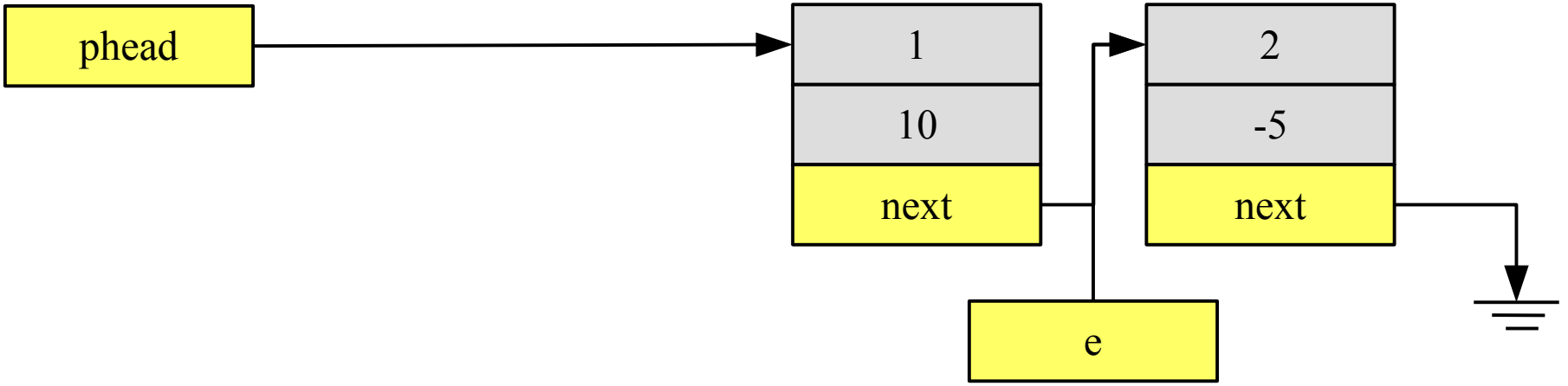
```
imprimeElemento(buscaCoord(2,-5,phead));
  struct coord * buscaCoord(int x, int y, struct coord * p)
  {
     if (p)
        if ((x==p->x) && (y==p->y))
           return p;
        else
           return buscaCoord(x, y, p->next);
     else
        return NULL;
  }
```



Buscando Elemento

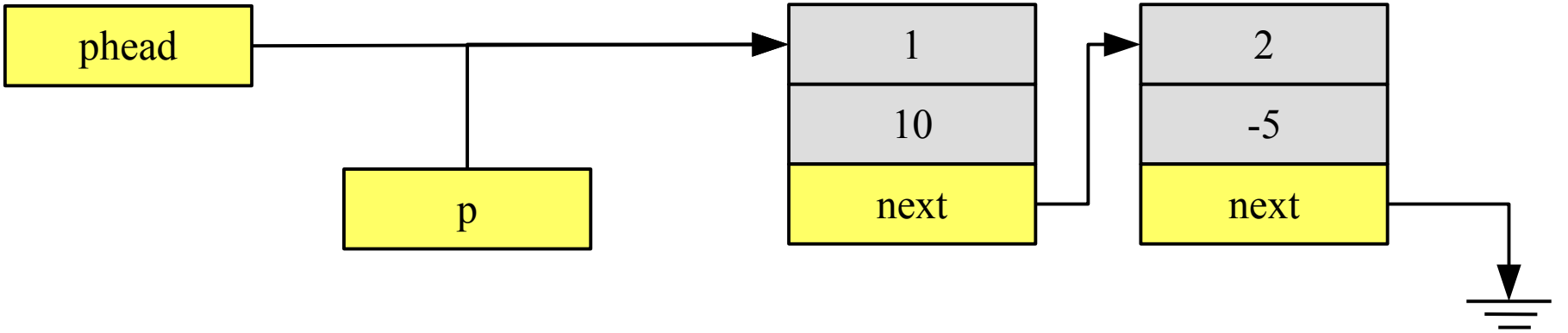
```
imprimeElemento(buscaCoord(2,-5,phead));
void imprimeElemento(struct coord * e)
{
   if (e)
      printf("(%hd,%hd)\n", e->x, e->y);
   else
      printf("Elemento inexistente!\n");
}
```





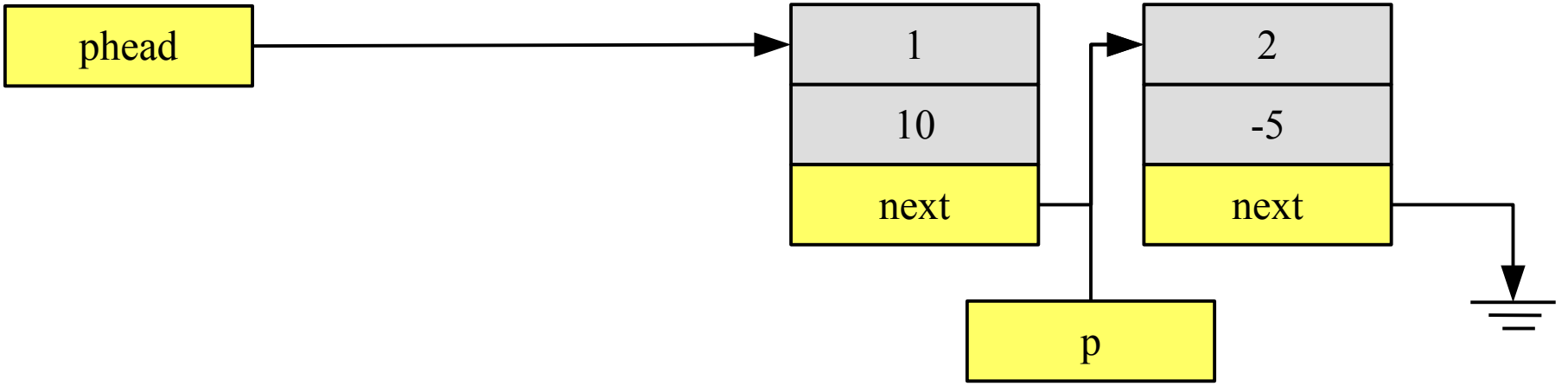
Buscando Elemento

```
imprimeElemento(buscaCoord(3,-5,phead));
  struct coord * buscaCoord(int x, int y, struct coord * p)
  {
     if (p)
        if ((x==p->x) && (y==p->y))
           return p;
        else
        return buscaCoord(x, y, p->next);
     else
        return NULL;
  }
```



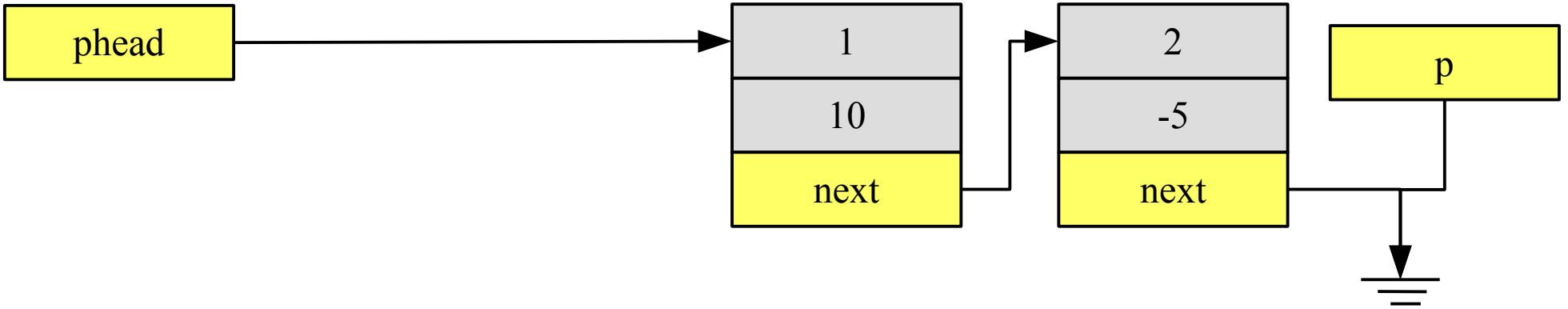
Buscando Elemento

```
imprimeElemento(buscaCoord(3,-5,phead));
  struct coord * buscaCoord(int x, int y, struct coord * p)
  {
     if (p)
        if ((x==p->x) && (y==p->y))
           return p;
        else
        return buscaCoord(x, y, p->next);
     else
        return NULL;
  }
```



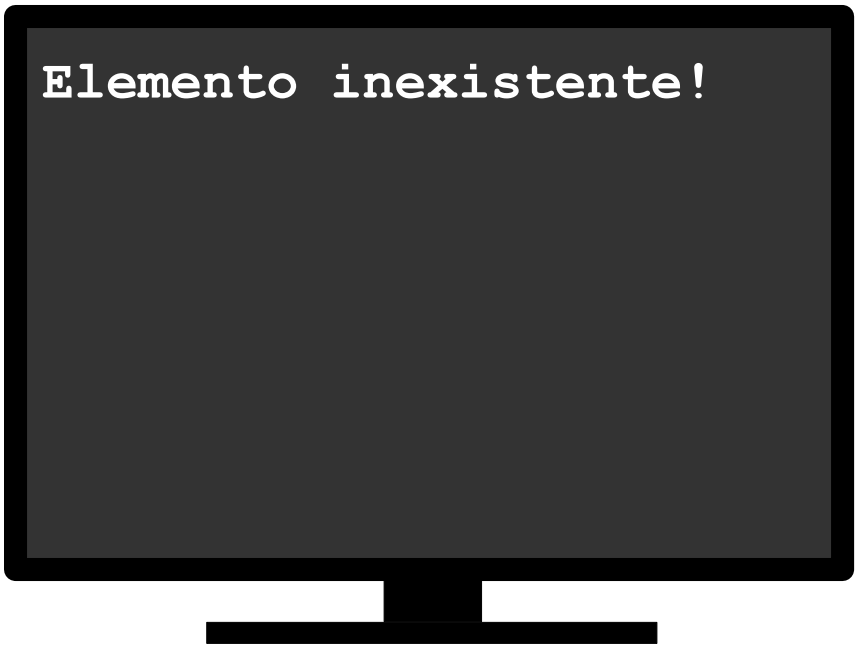
Buscando Elemento

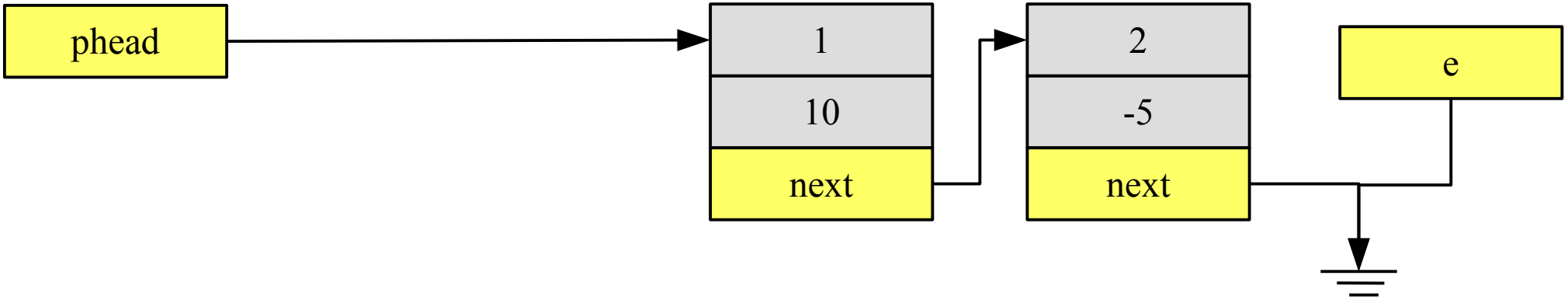
```
imprimeElemento(buscaCoord(3,-5,phead));
  struct coord * buscaCoord(int x, int y, struct coord * p)
  {
     if (p)
        if ((x==p->x) && (y==p->y))
           return p;
        else
           return buscaCoord(x, y, p->next);
     else
  ➡  return NULL;
  }
```



Buscando Elemento

```
imprimeElemento(buscaCoord(3,-5,phead));
void imprimeElemento(struct coord * e)
{
   if (e)
      printf("(%hd,%hd)\n",e->x, e->y);
   else
      printf("Elemento inexistente!\n");
}
```





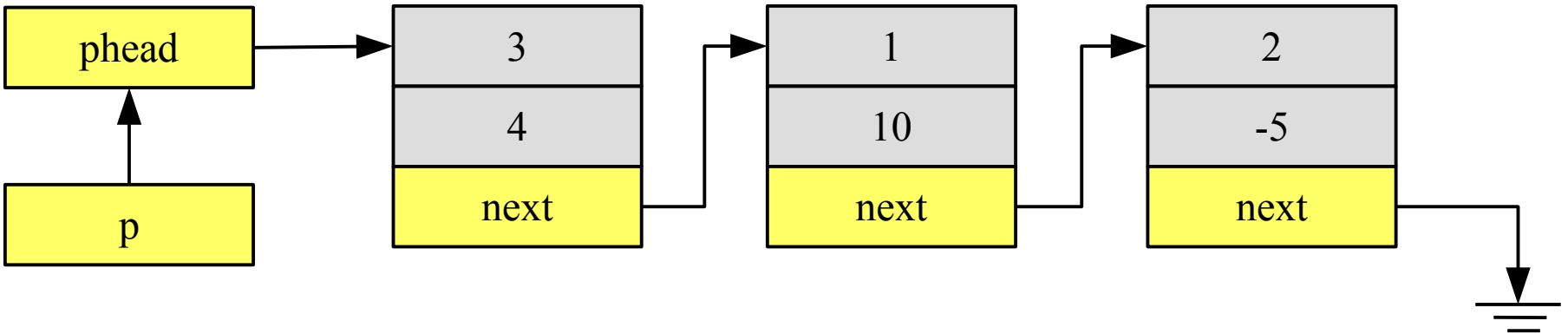
Exclusão em Listas Encadeadas

Excluindo Elemento (pelas coordenas)

```
void excluiElementoCoord(int x, int y, struct coord ** p)
{
   if (*p) {

      if ((x==(*p)->x) && (y==(*p)->y)) {

         struct coord * e = *p;
         *p = e->next;
         free(e);
         printf("Elemento excluído com sucesso!\n");
      }
      else
         excluiElementoCoord(x,y, &((*p)->next));
   }
   else
      printf("Elemento não encontrado!\n");
}
```

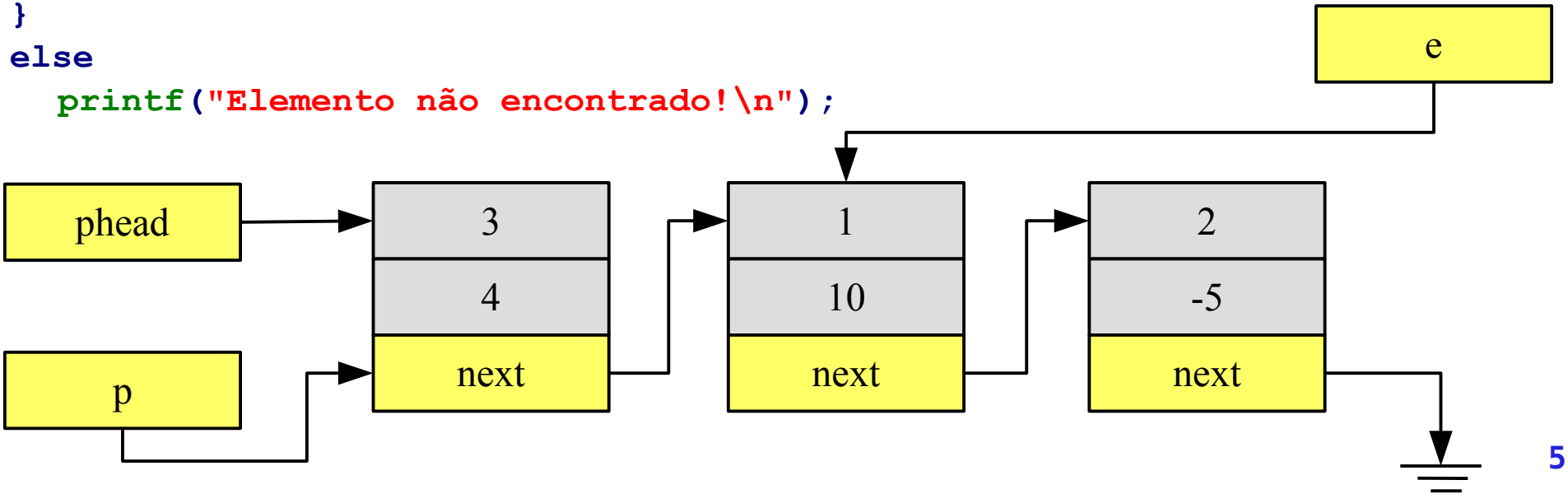
Excluindo Elemento (pelas coordenas)

```
excluiElementoCoord(1,10,&phead);
void excluiElementoCoord(int x, int y, struct coord ** p)
{
   if (*p) {

      if ((x==(*p)->x) && (y==(*p)->y)) {

         struct coord * e = *p;
         *p = e->next;
         free(e);
         printf("Elemento excluído com sucesso!\n");
      }
      else
         excluiElementoCoord(x,y, &((*p)->next));
   }
   else
      printf("Elemento não encontrado!\n");
}
```



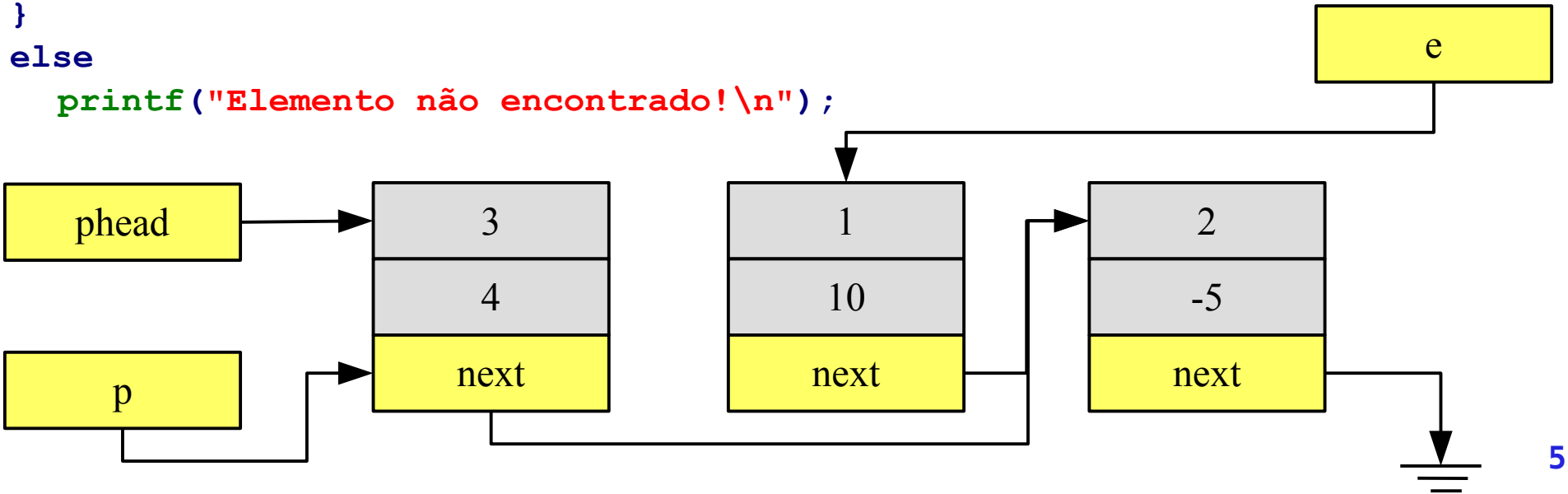
Excluindo Elemento (pelas coordenas)

```
excluiElementoCoord(1,10,&phead);
void excluiElementoCoord(int x, int y, struct coord ** p)
{
   if (*p) {

      if ((x==(*p)->x) && (y==(*p)->y)) {

         struct coord * e = *p;
         *p = e->next;
         free(e);
         printf("Elemento excluído com sucesso!\n");
      }
      else
         excluiElementoCoord(x,y, &((*p)->next));
   }
   else
      printf("Elemento não encontrado!\n");
}
```



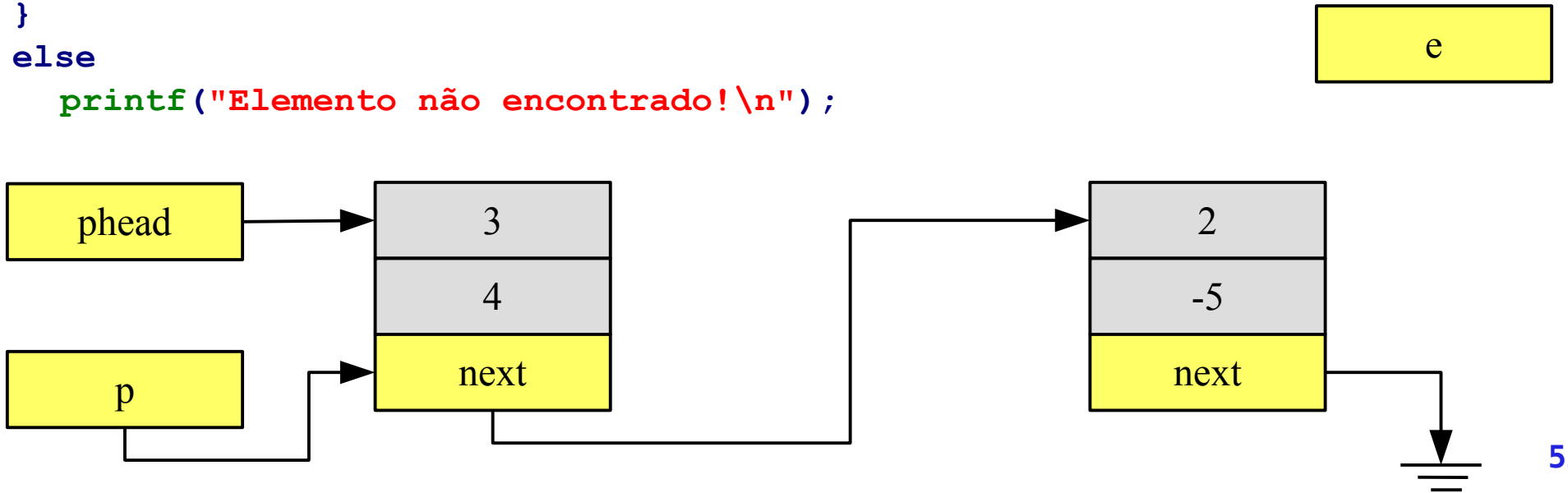
Excluindo Elemento (pelas coordenas)

```
excluiElementoCoord(1,10,&phead);
void excluiElementoCoord(int x, int y, struct coord ** p)
{
   if (*p) {

      if ((x==(*p)->x) && (y==(*p)->y))   {

         struct coord * e = *p;
         *p = e->next;
         free(e);
         printf("Elemento excluído com sucesso!\n");
      }
      else
         excluiElementoCoord(x,y, &((*p)->next));
   }
   else
      printf("Elemento não encontrado!\n");
}
```



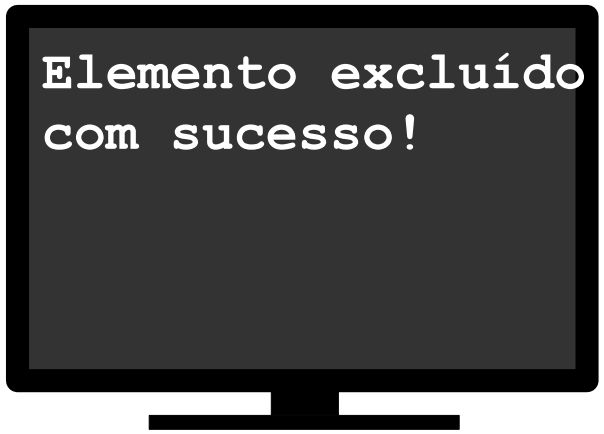
Excluindo Elemento (pelas coordenas)

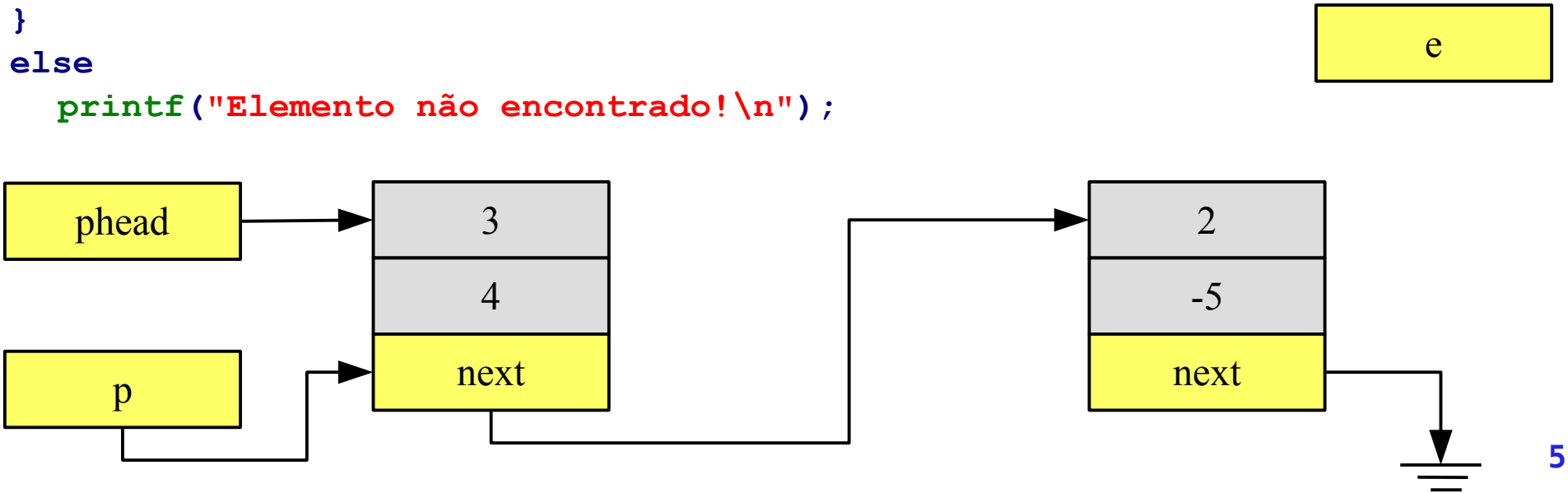
```
excluiElementoCoord(1,10,&phead);
void excluiElementoCoord(int x, int y, struct coord ** p)
{
   if (*p) {

      if ((x==(*p)->x) && (y==(*p)->y)) {

         struct coord * e = *p;
         *p = e->next;
         free(e);
         printf("Elemento excluído com sucesso!\n");
      }
      else
         excluiElementoCoord(x,y, &((*p)->next));
   }
   else
      printf("Elemento não encontrado!\n");
}
```



Excluindo Elemento (pelas coordenas)

```
excluiElementoCoord(1,10,&phead);
void excluiElementoCoord(int x, int y, struct coord ** p)
{
   if (*p)
   {
      if ((x==(*p)->x) && (y==(*p)->y))
      {
         struct coord * e = *p;
         *p = e->next;
         free(e);
         printf("Elemento excluído com sucesso!\n");
      }
      else
         excluiElementoCoord(x,y, &((*p)->next));
   }
   else
      printf("Elemento não encontrado!\n");
}
```



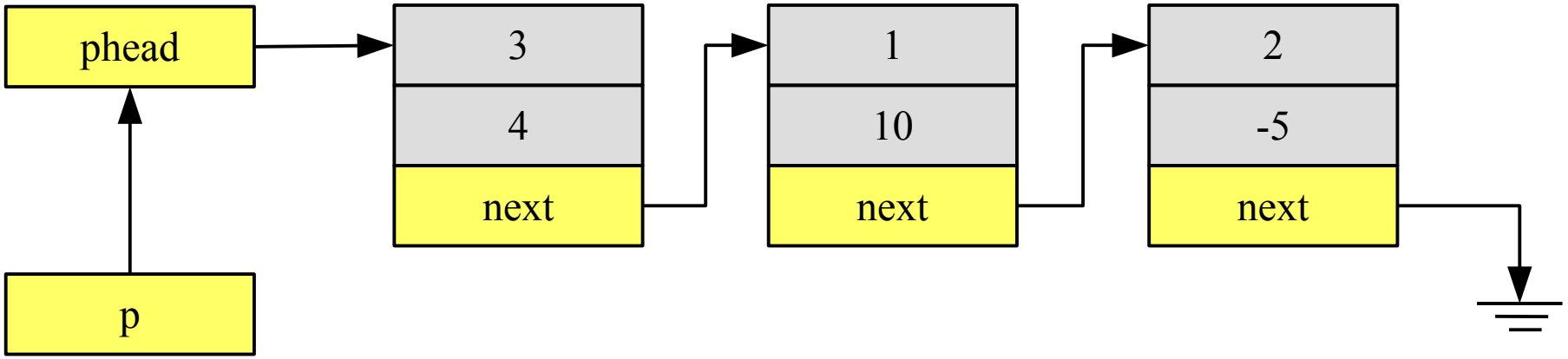


Excluindo Elemento (pelo apontador)

```
void excluiElementoPonteiro(struct coord * e, struct coord
** p) {

    if (*p) {

        if (*p == e)  {

            *p = e->next;
            free(e);
            printf("Elemento excluído com sucesso!\n");
        }
        else
            excluiElementoPonteiro(e, &((*p)->next));
    }
    else
        printf("Elemento não encontrado!\n");
}
```

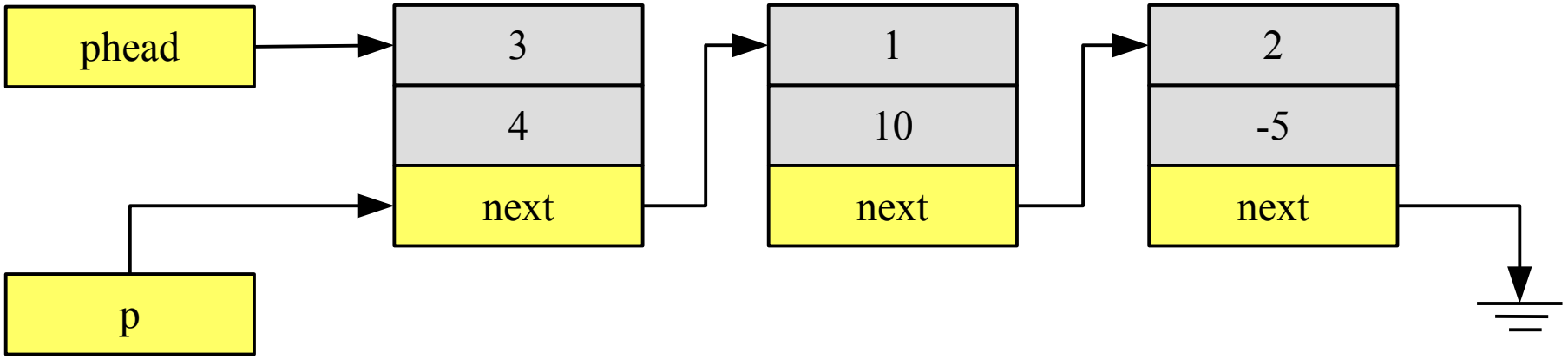
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



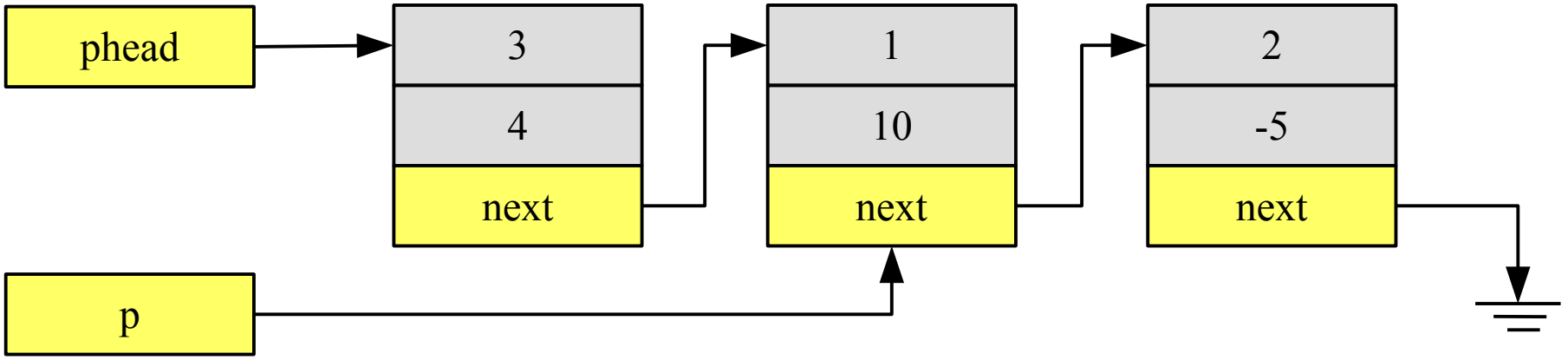
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



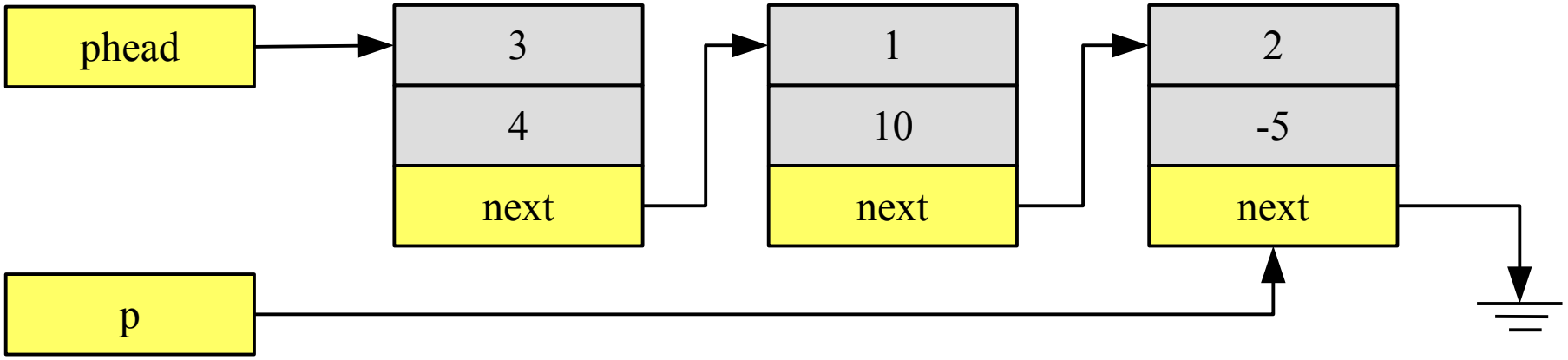
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

     if (*p) {
          limparLista(&((*p)->next))
          free(*p);
          *p = NULL;
     }
}
```



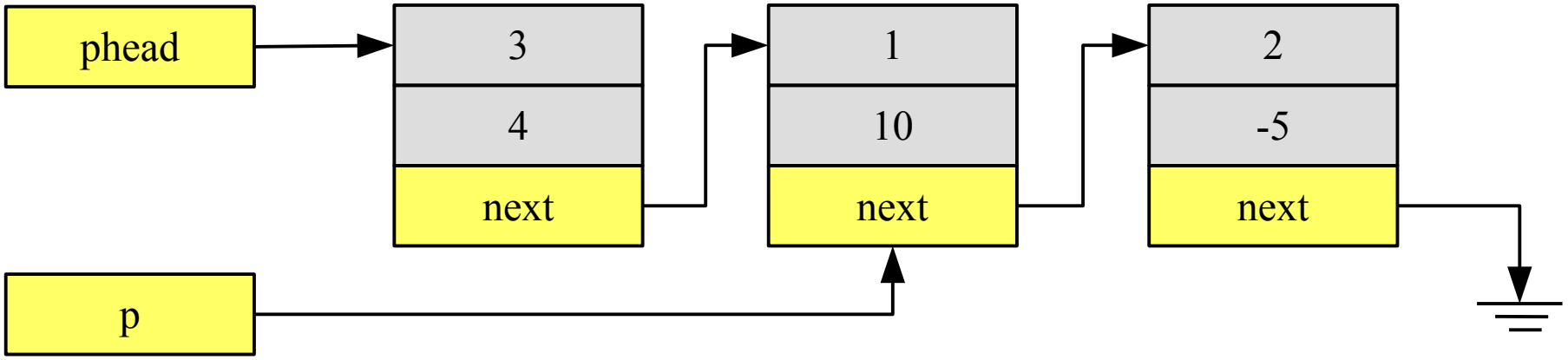
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



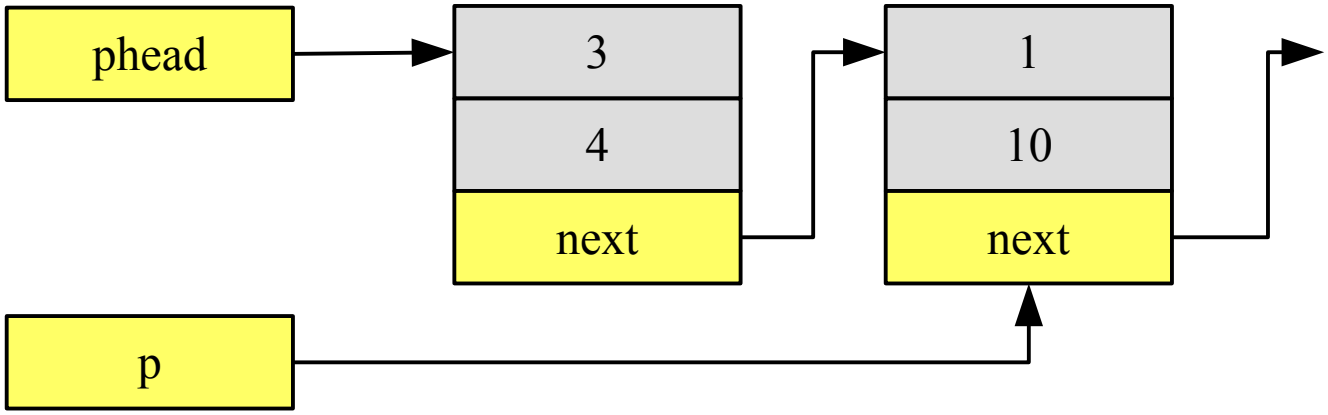
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



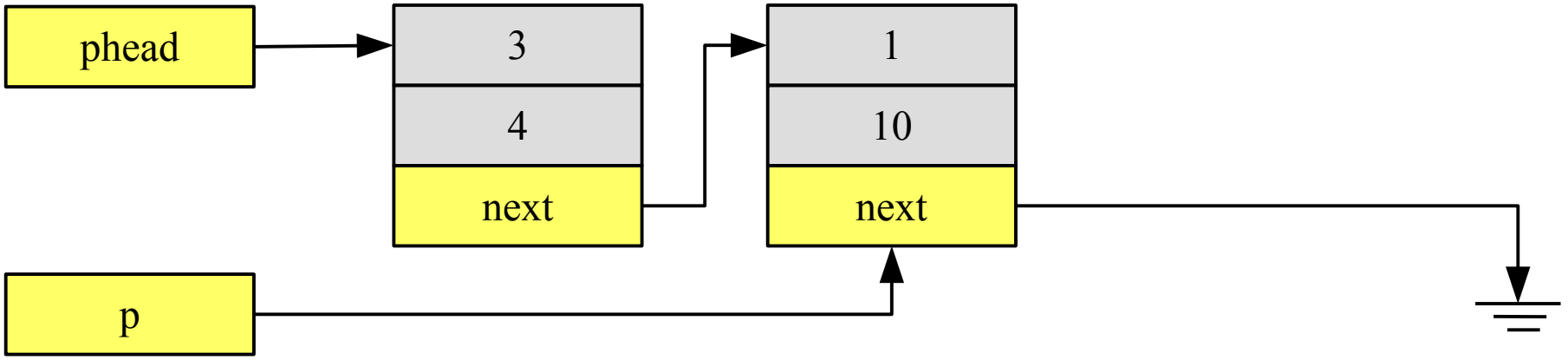
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



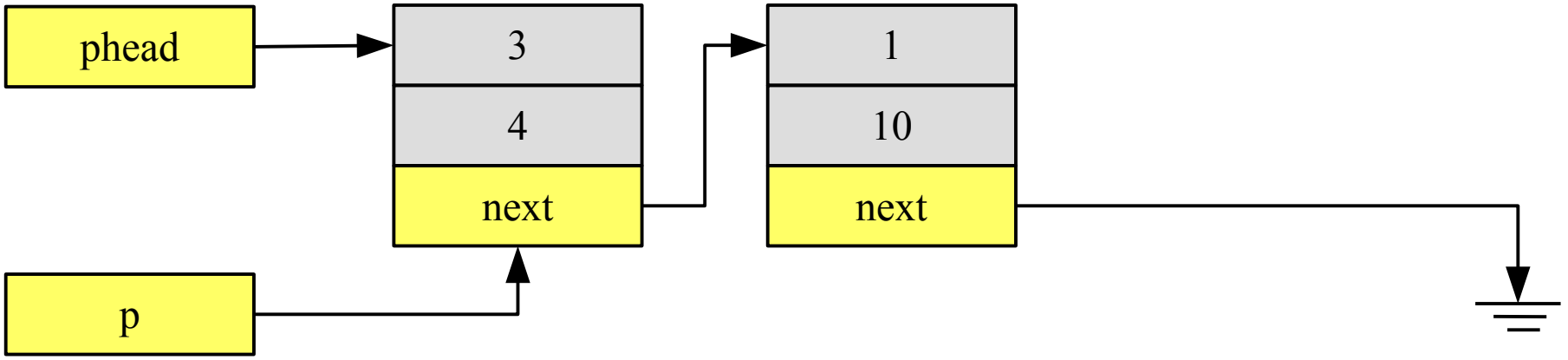
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p){

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



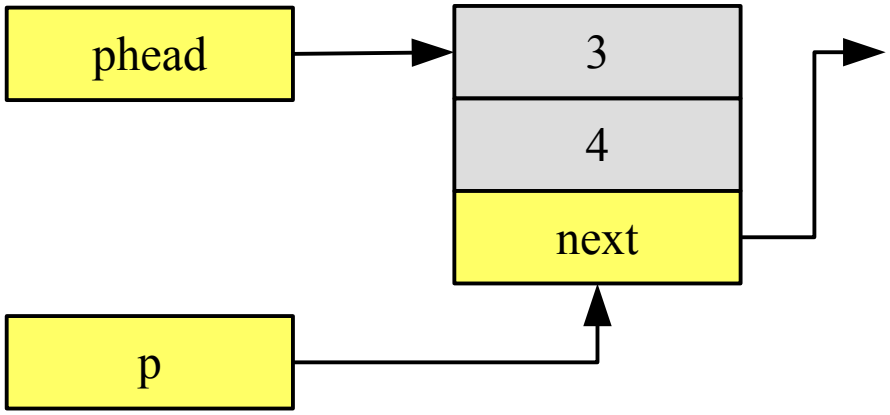
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



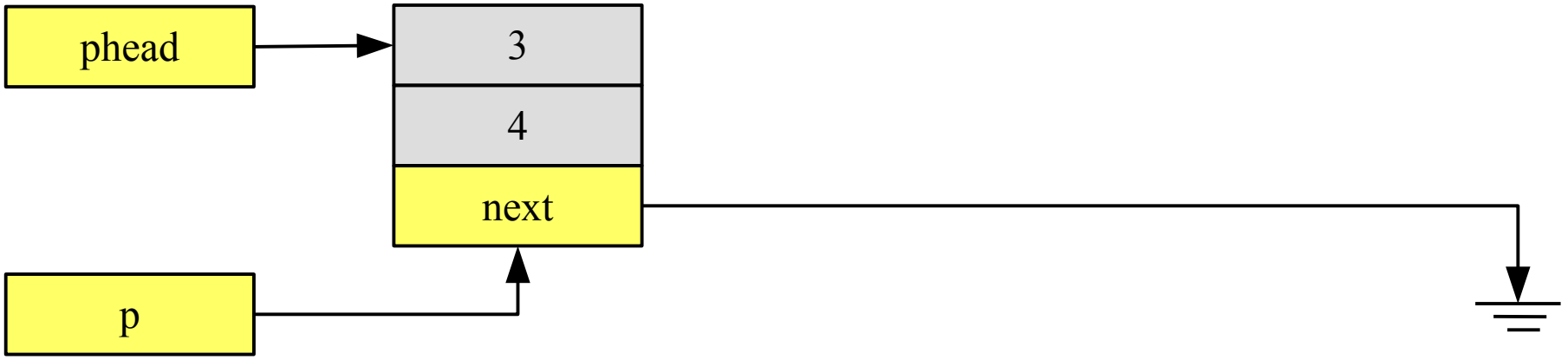
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p)  {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



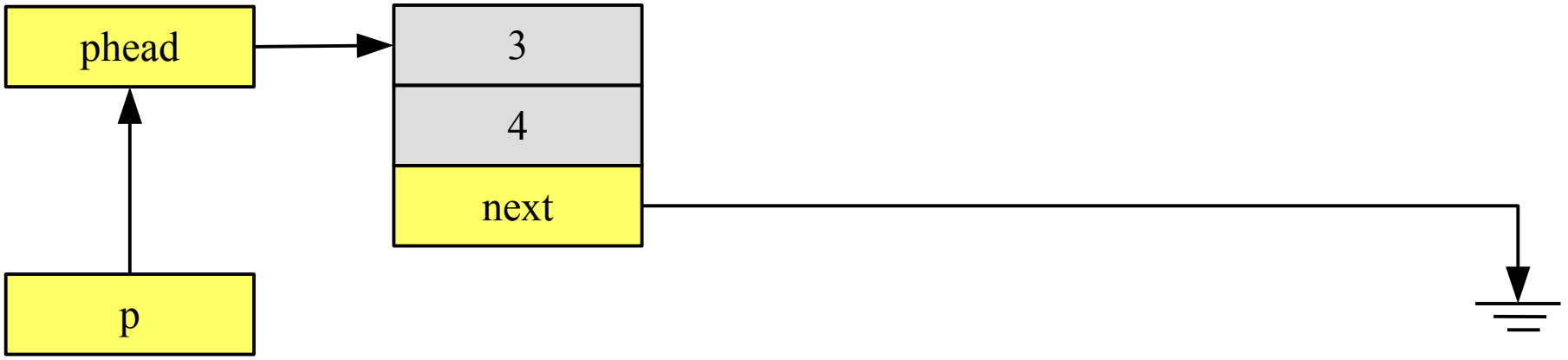
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p)  {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



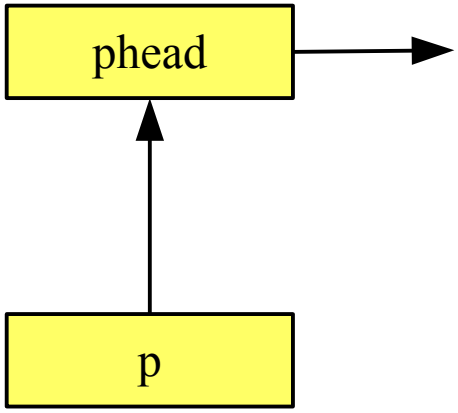
Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p){

    if (*p) {

        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p) {

    if (*p)
    {
        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)

void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        free(*p);
        *p = NULL;
    }
}
```

Usando isso tudo!

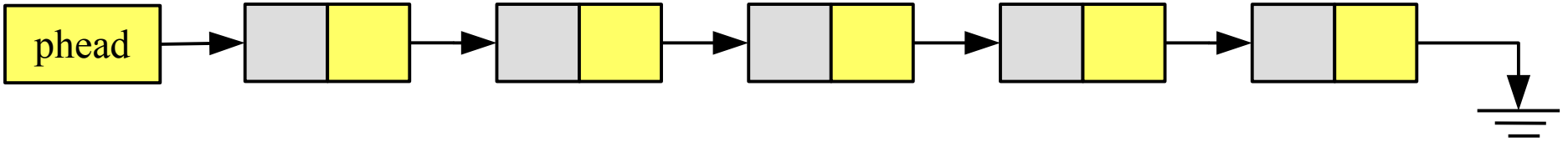
```
int main()
{
    struct coord * phead = NULL;
    int i;
    //Insere 3 elementos no final da lista
    for (i=0; i<3; i++)
        insereFim(&phead);

    //insere 2 elementos no inicio da lista
    for (i=0; i<2; i++)
        insereInicio(&phead);

    imprimeLista(phead);

    //Busca e imprime o elemento (1,10)
    imprimeElemento(buscaCoord(1, 10, phead));
```

Usando isso tudo!

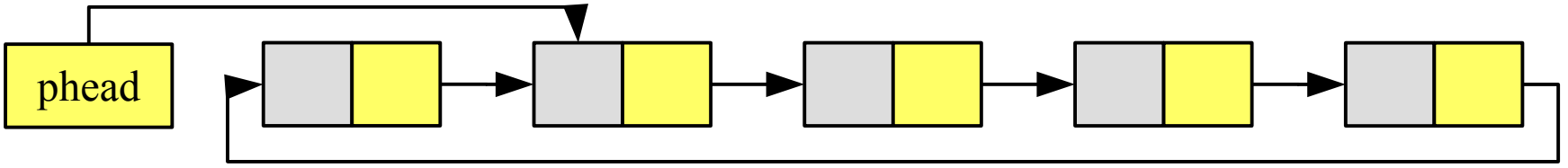
```
    /*Busca e imprime o elemento (2,10);*/
    imprimeElemento(buscaCoord(2, 10, phead));

    excluiElementoCoord(1, 10, &phead);
    imprimeLista(phead);

    excluiElementoPonteiro(buscaCoord(2, 20, phead),
                           &phead);
    imprimeLista(phead);

    excluiElementoPonteiro(buscaCoord(2, 20, phead),
                           &phead);

    limparLista(&phead);
    imprimeLista(phead);
}
```

Alguns tipos de lista encadeada

- Lista encadeada simples.

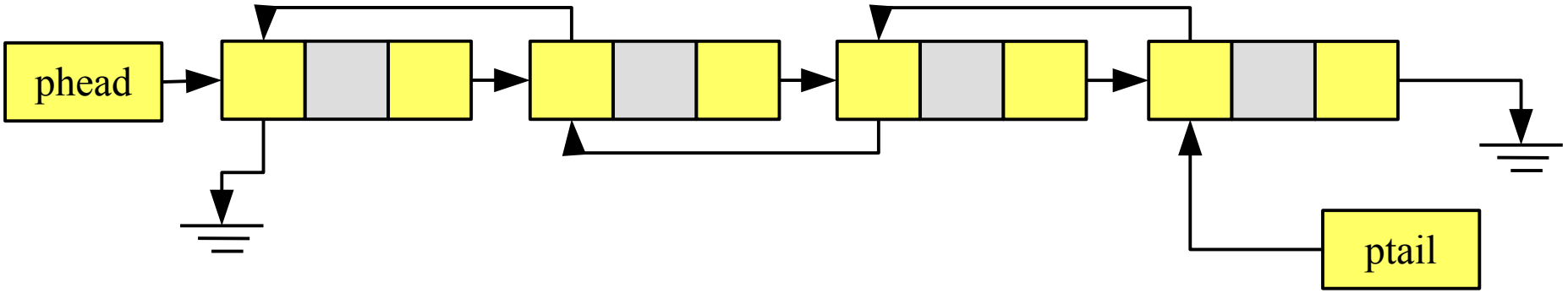


- Lista encadeada circular
  - phead pode apontar para qualquer elemento, pois a lista não tem início.

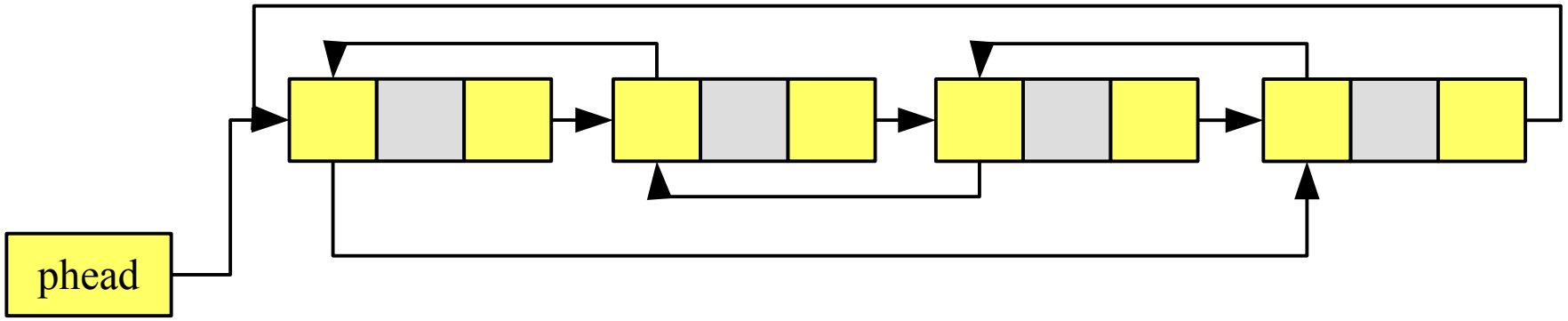


Alguns tipos de lista encadeada

- Lista duplamente encadeada com sentinelas.



- Lista duplamente encadeada circular.
  - phead pode apontar para qualquer elemento, pois a lista não tem início



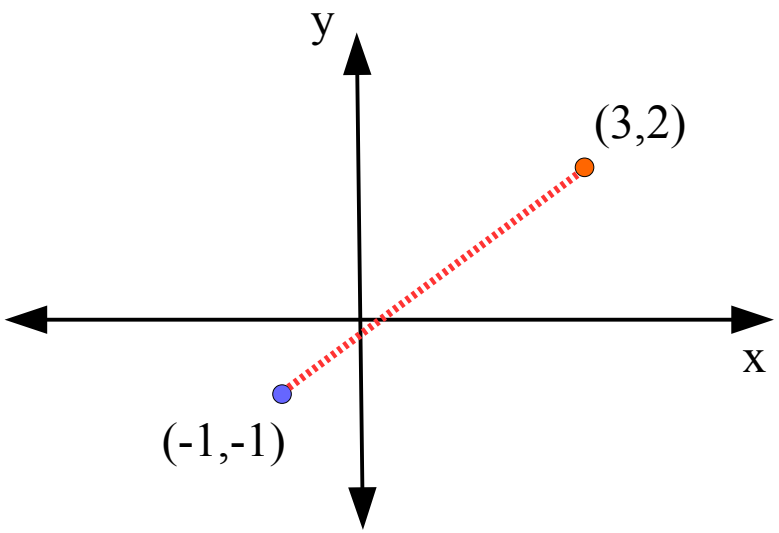
Exercícios

Exercício

- Criar 3 novas funções no projeto da lista de coordenadas:
  - Função para ordenar a lista pela distância euclidiana entre o ponto e a origem (0,0). Chame a função de `ordenaListaEuclidiana`.
  - Função para inserir um novo ponto em uma lista ordenada (também pela distância euclidiana entre o ponto e a origem). Chame a função de `insereOrdenadoEuclidiana`.
  - Modificar as funções de exclusão para imprimir o elemento a ser excluído e pedir uma confirmação antes de realmente apagar.

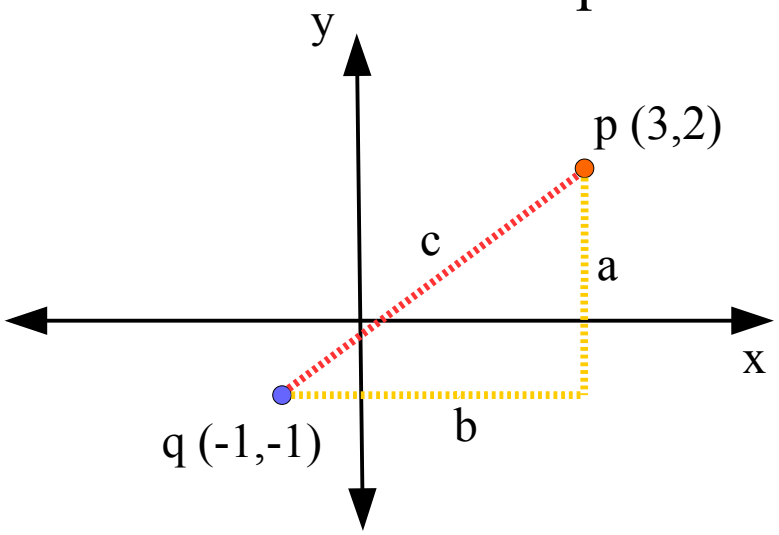
Exercício

- O que é distância euclidiana?
  - No 2D



Exercício

- O que é distância euclidiana?
  - No 2D
    - Basta marcar o triângulo retângulo e calcular a hipotenusa.
    - Cateto a = 2 - (-1) = 3
    - Cateto b = 3 - (-1) = 4
    - Hipotenusa c = 5 (pitagoras)



Em geral, temos:

$$DistEuc_{p \to q} = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$