

Unidade VIII - Estruturas autorreferenciadas

Disciplina Linguagens de Programação I
Bacharelado em Ciência da Computação da Uerj
Professores Guilherme Mota e Leandro Marzulo

ANSI 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;

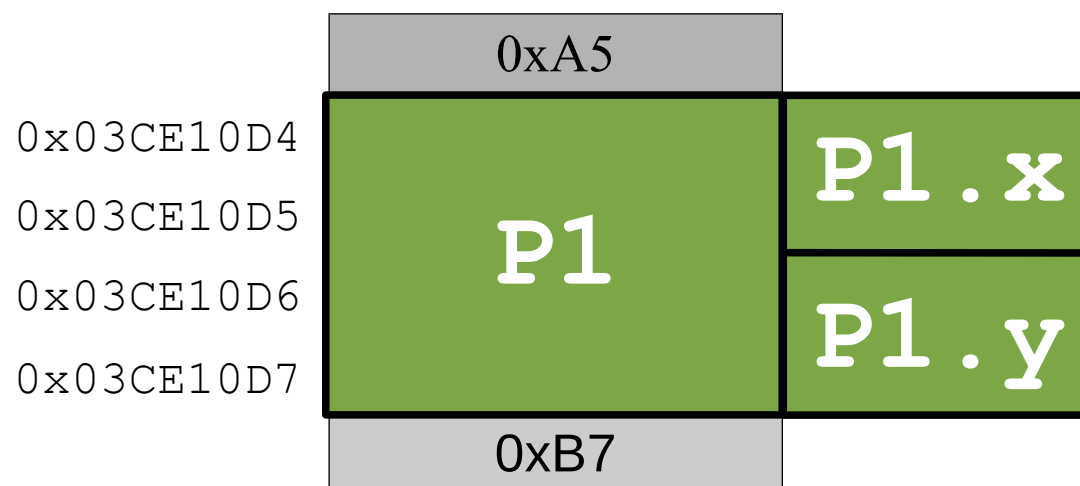
...
clist =(struct coord *) malloc (2*sizeof(struct coord));
int i;
for(i=0; i<2; i++)
{
    clist[i].x = i;
    (*(clist+i)).y = 2 * i;
    printf("X = (%d, %d) \n", (clist+i)->x, (clist+i)->y);
}
free(clist);
```

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;
```

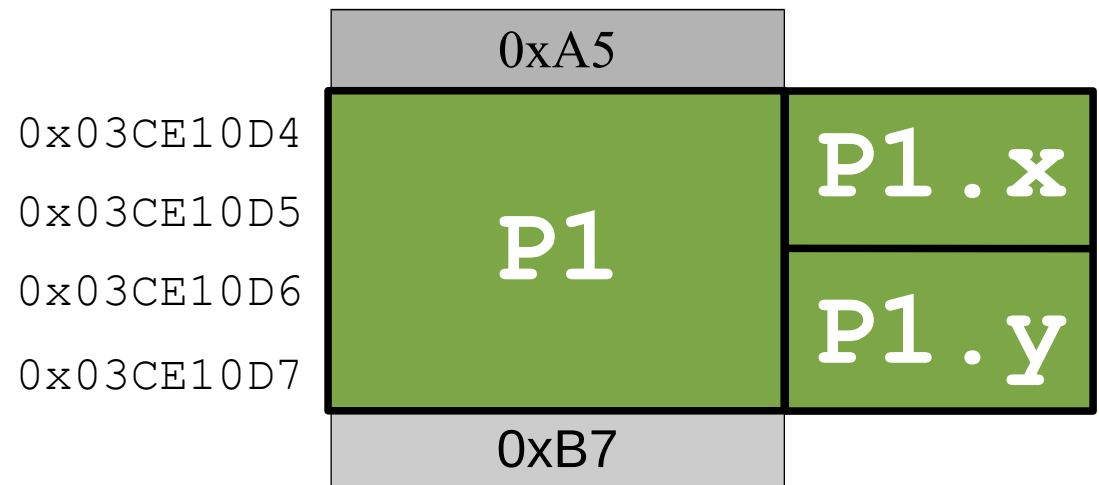


Operador . Vs Operador ->

- 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;
```

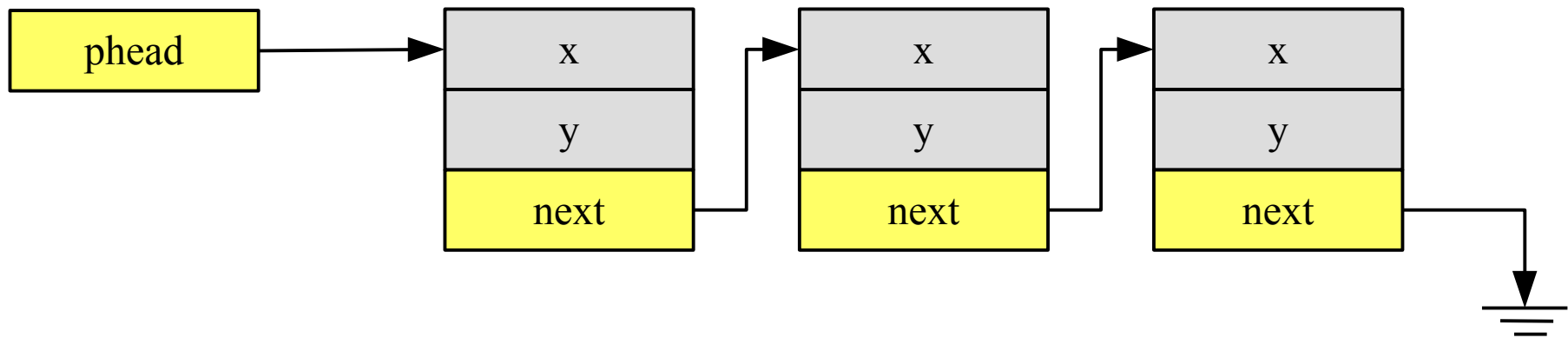
```
(* (Vetor+3)).y = 10;
```

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


Listas Encadeadas

O que é uma lista encadeada?

- É uma lista cujos elementos estão espalhados em qualquer lugar da memória (em qualquer ordem).
- Temos um apontador para o primeiro elemento da lista
- Cada elemento da lista aponta para o seguinte.



Visão de uma lista encadeada na memória

0x0000000003CE10CB	0x3CE10DB	p h e a d
0x0000000003CE10CC		
0x0000000003CE10CD		
0x0000000003CE10CE		
0x0000000003CE10CF		
0x0000000003CE10D0		
0x0000000003CE10D1		
0x0000000003CE10D2	0xA5	n e x t
0x0000000003CE10D3		
0x0000000003CE10D4		
0x0000000003CE10D5		
0x0000000003CE10D6		
0x0000000003CE10D7		
0x0000000003CE10D8		
0x0000000003CE10D9	0x96	0xB7
0x0000000003CE10DA		
	0xB7	0x32
	0x32	0x08
	0x08	

0x0000000003CE10DB	0xA5	x
0x0000000003CE10DC	0xFD	
0x0000000003CE10DD	0x01	y
0x0000000003CE10DE	0xFF	
0x0000000003CE10DF	0x4FFFFCA	n e x t
0x0000000003CE10E0		
0x0000000003CE10E1		
0x0000000003CE10E2		
0x0000000003CE10E3		
0x0000000003CE10E4		
0x0000000003CE10E5		
0x0000000003CE10E6	0x96	0xB7
0x0000000003CE10E7		
0x0000000003CE10E8		
0x0000000003CE10E9	0x32	0x08
0x0000000003CE10EA		
	0x08	

Algumas vantagens do uso de listas encadeadas

- Realloc é custoso para aumentar o tamanho de estruturas dinamicamente, pois envolve cópia de dados.
- Com listas encadeadas podemos adicionar e remover elementos sem precisar copiar os demais elementos para outros locais.
- No caso de exclusão de elementos, podemos marcar o elemento como livre para evitar a movimentação dentro de vetores, mas no caso da inclusão, o realloc seria inevitável.
- Operações de ordenação se tornam muito mais baratas, pois os dados não são movimentados, sendo apenas necessário ajustar os apontadores.

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

Inserção em Listas Encadeadas

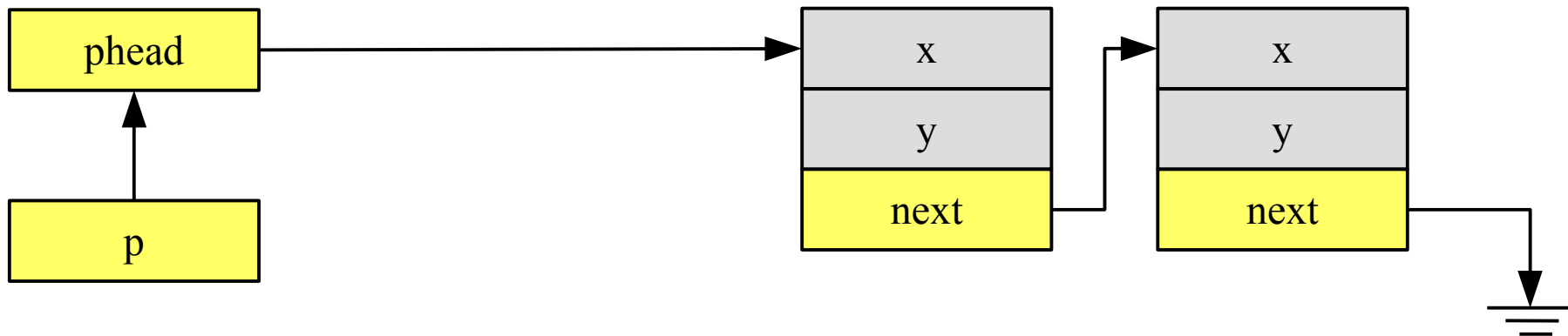
Inserindo um elemento no inicio da lista

```
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) ;
```

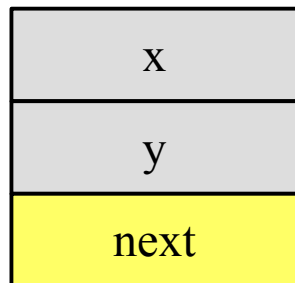
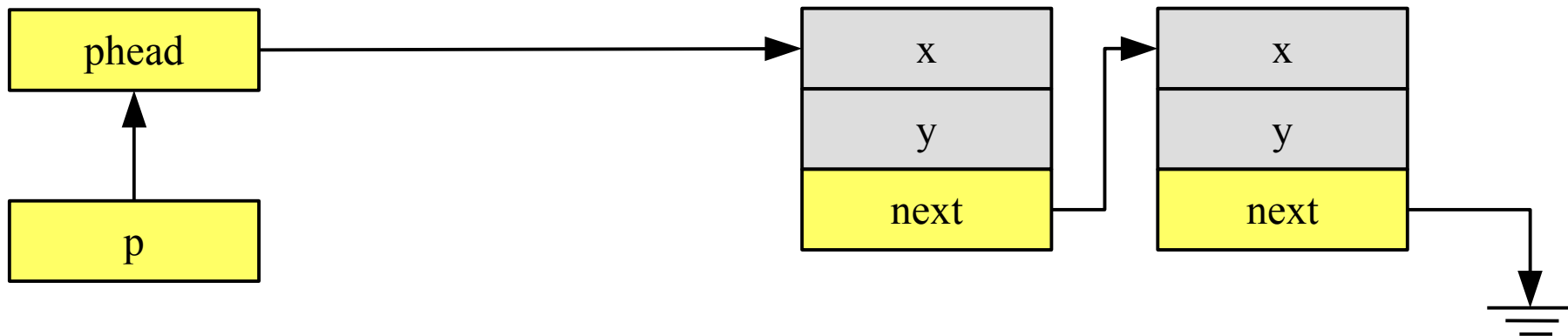
```
void insereInicio(struct coord ** p, short x, short y)
{
    struct coord * elem = criaElem(x,y) ;
    elem->next= *p;
    *p=elem;
}
```



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;
}
```

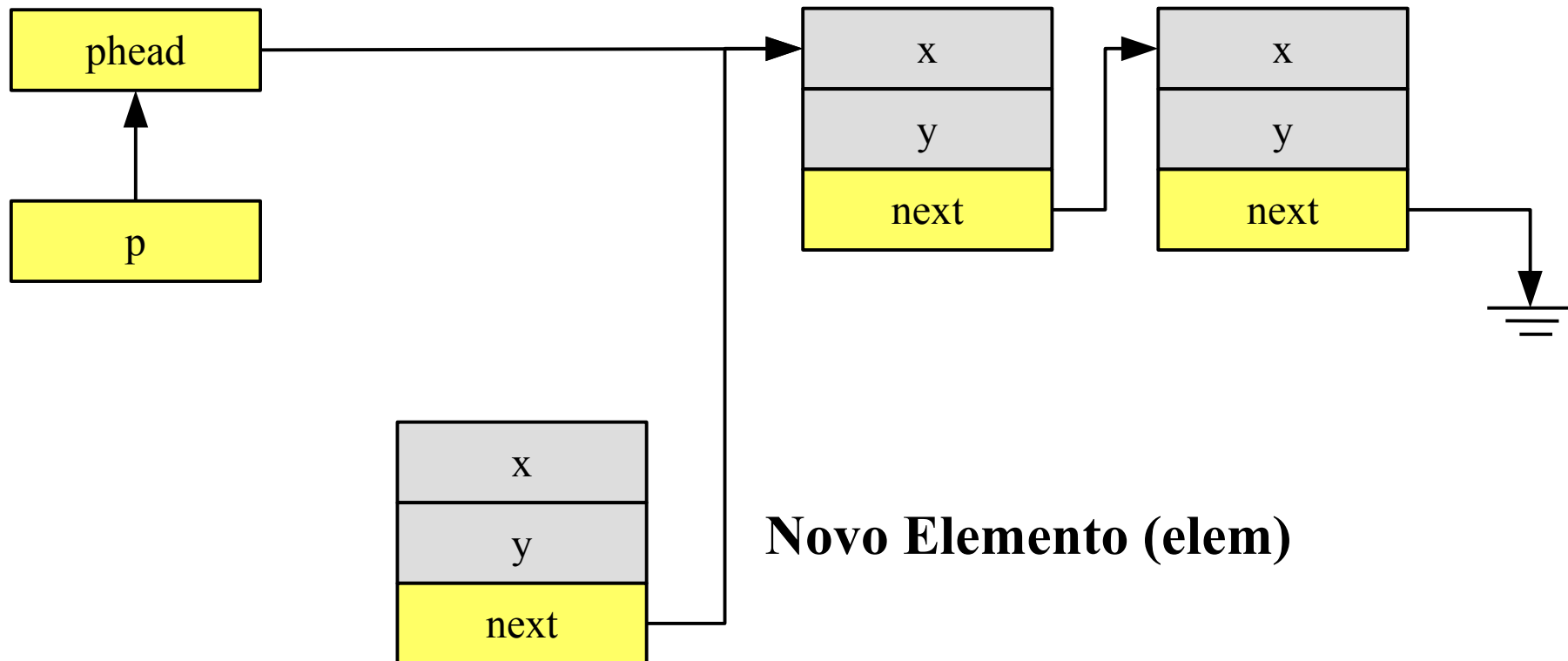


Novo Elemento (elem)

Inserindo um elemento no inicio da lista

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

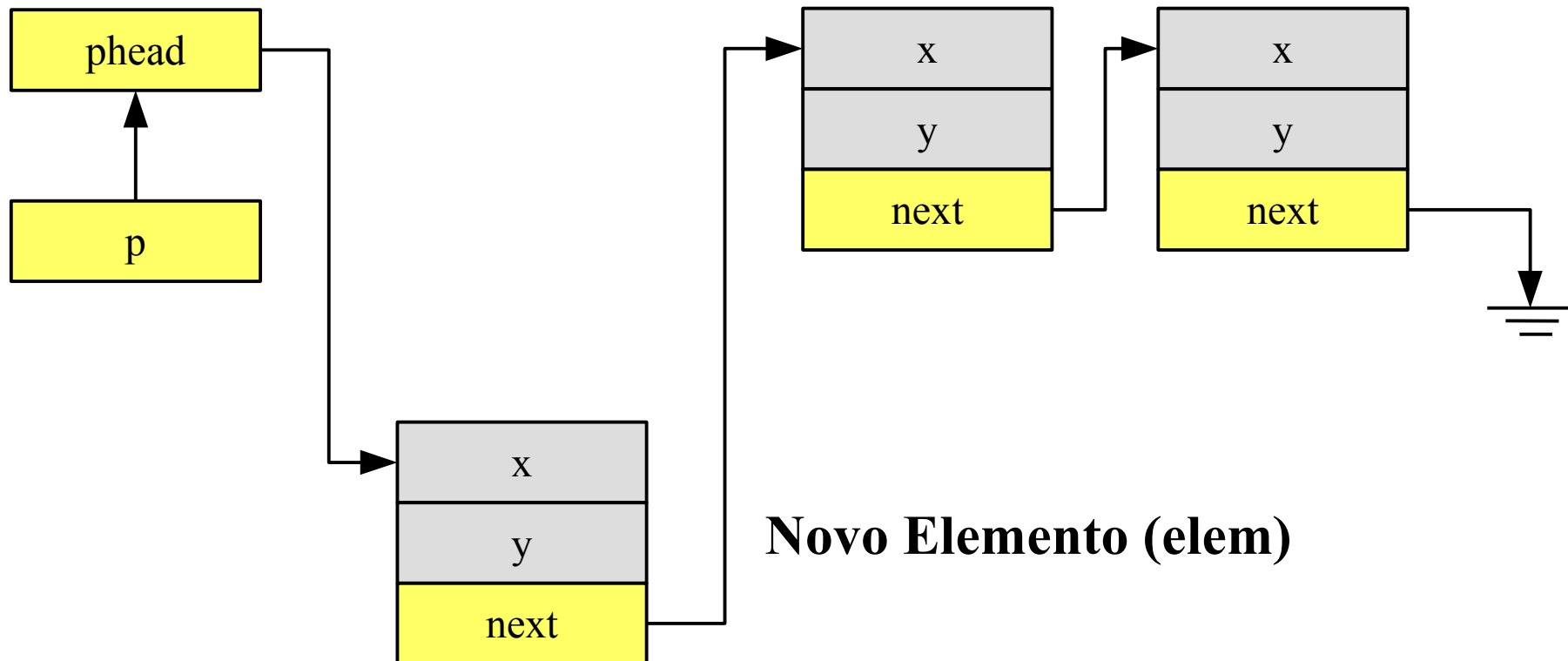
```
void insereInicio(struct coord ** p, short x, short y)
{
    struct coord * elem = criaElem(x,y) ;
    elem->next= *p;
    *p=elem;
}
```



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;
}
```



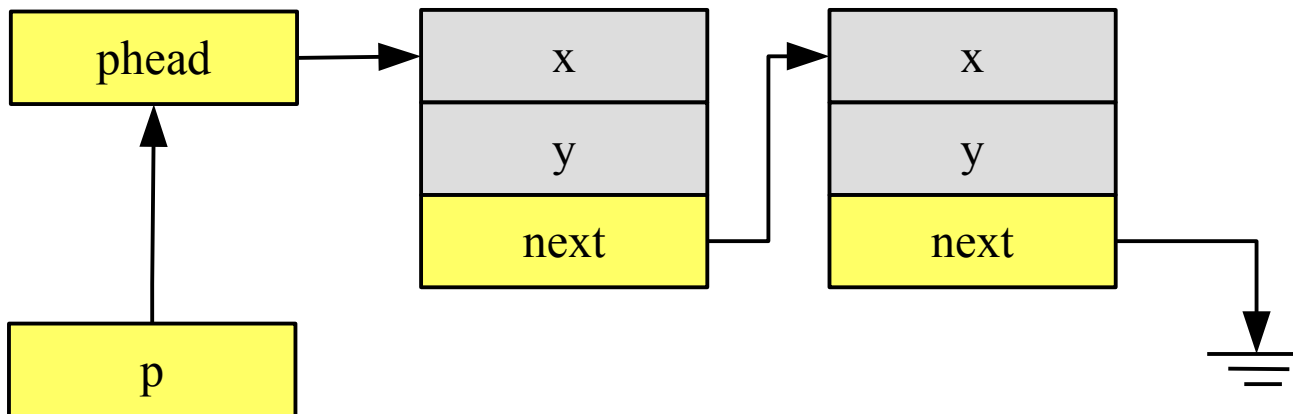
Inserindo um elemento no final da lista

```
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) ;
```

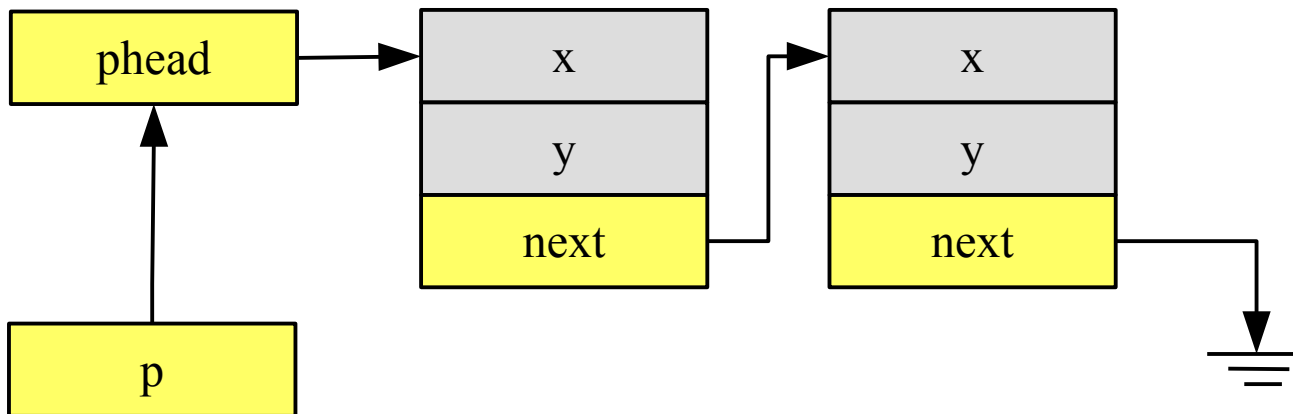
```
void insereFim(struct coord ** p, short x, short y)
{
    if (*p)
        insereFim(&((*p)->next), x, y);
    else
    {
        struct coord * elem = criaElem(x,y);
        elem->next= NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

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

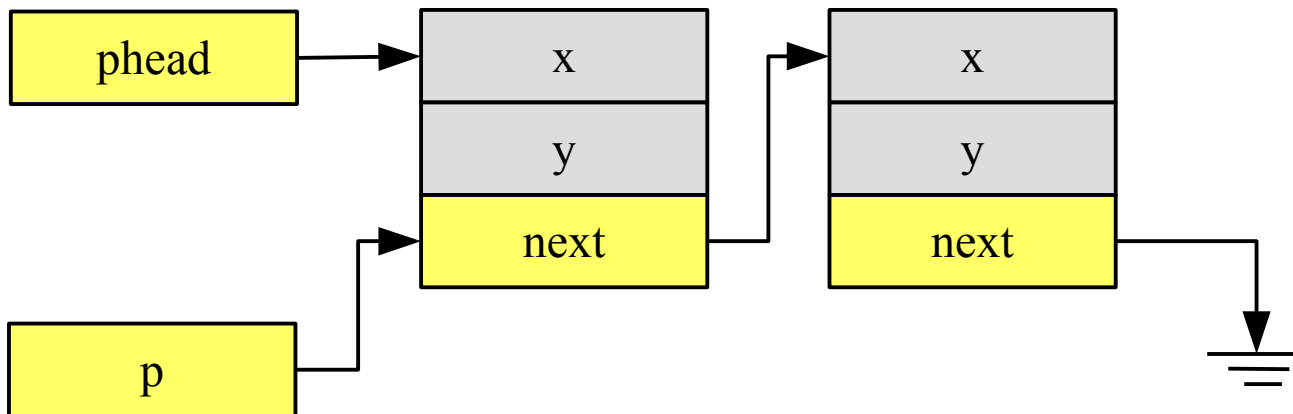
```
void insereFim(struct coord ** p, short x, short y)
{
    if (*p)
        → insereFim(&((*p)->next), x, y);
    else
    {
        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

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

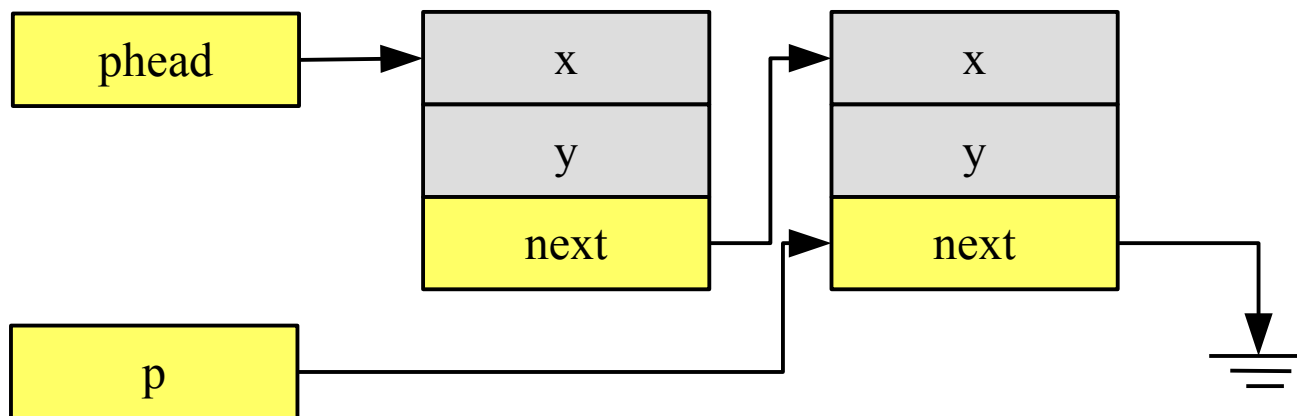
```
void insereFim(struct coord ** p, short x, short y)
{
    if (*p)
        → insereFim(&((*p)->next), x, y);
    else
    {
        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



Inserindo um elemento no final da lista

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

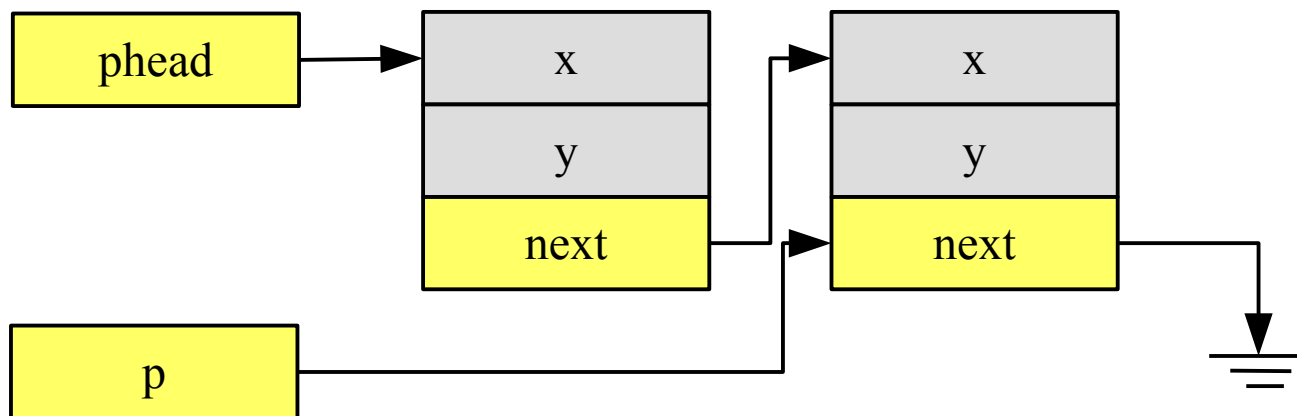
```
void insereFim(struct coord ** p, short x, short y)
{
    if (*p)
        insereFim(&((*p)->next), x, y);
    → else
    {
        struct coord * elem = criaElem(x, y);
        elem->next= NULL;
        *p=elem;
    }
}
```



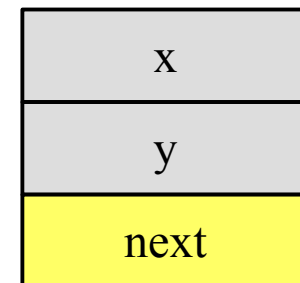
Inserindo um elemento no final da lista

```
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;
    }
}
```



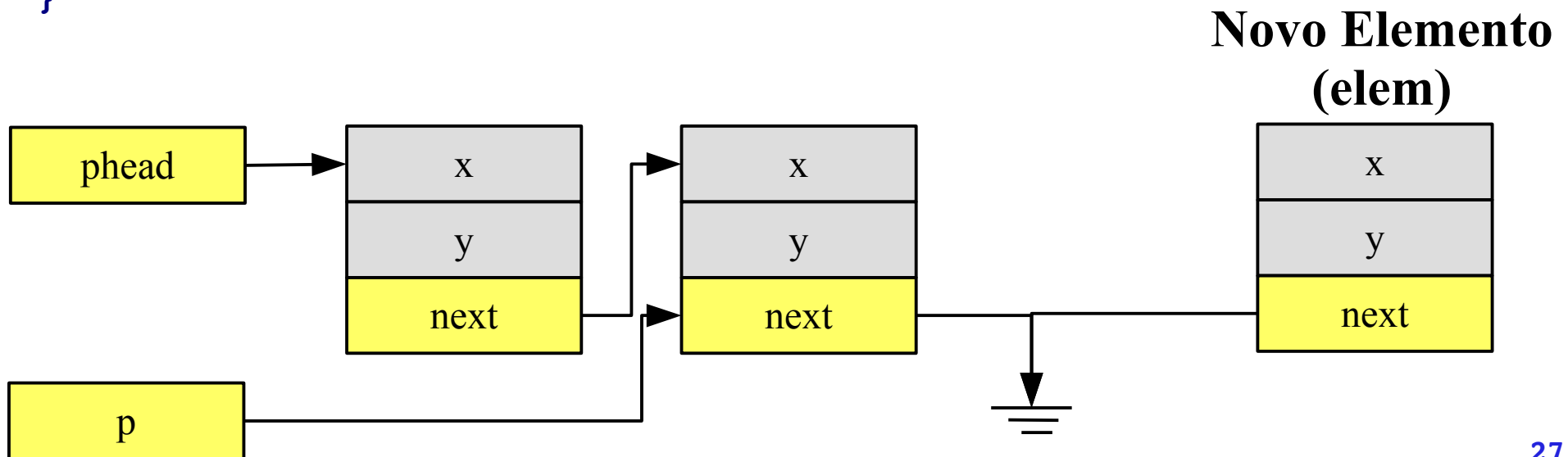
**Novo Elemento
(elem)**



Inserindo um elemento no final da lista

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

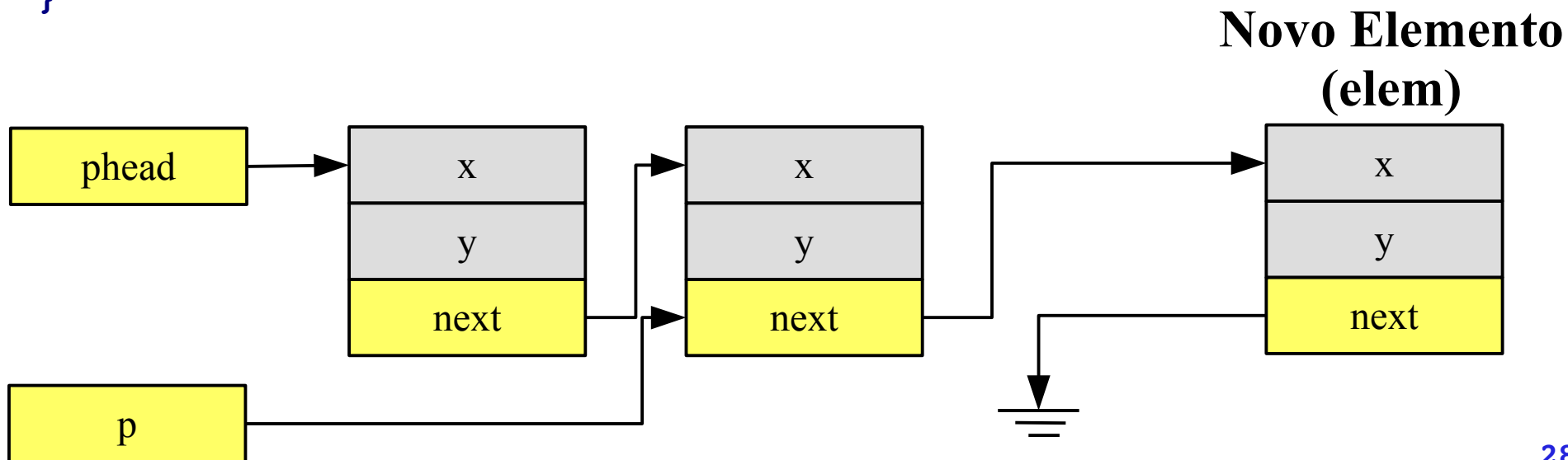
```
void insereFim(struct coord ** p)
{
    if (*p)
        insereFim(&((*p)->next));
    else
    {
        struct coord * elem = criaElem();
        ➔ elem->next = NULL;
        *p=elem;
    }
}
```



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

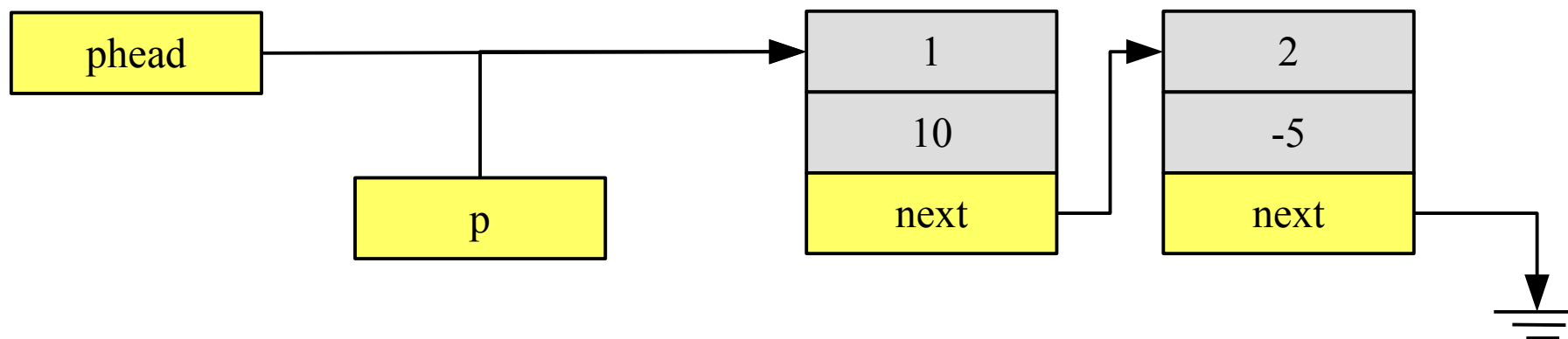
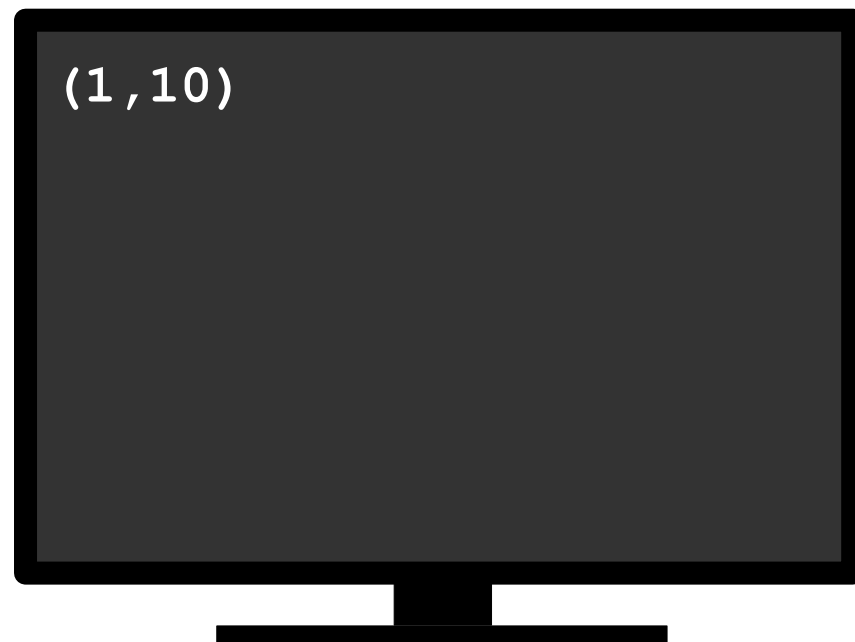
```
void imprimeElemento(struct coord * e)
{
    if (e)
        printf("( %hd, %hd) \n", e->x, e->y);
    else
        printf("Elemento inexistente! \n");
}
```

Imprimindo a lista

```
void imprimeLista (struct coord * p)
{
    if (p)
    {
        imprimeElemento (p) ;
        imprimeLista (p->next) ;
    }
    else
        printf("Fim da Lista\n") ;
}
```

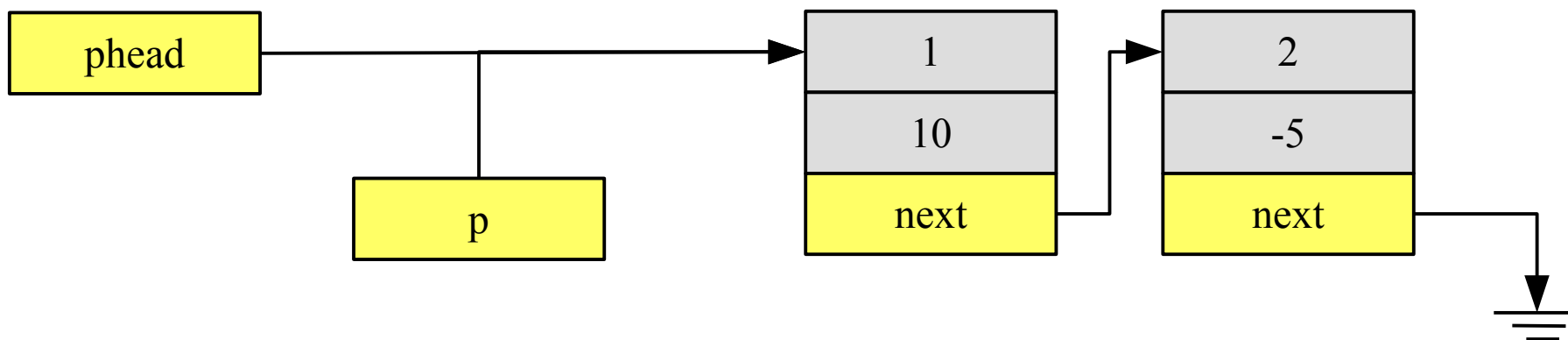
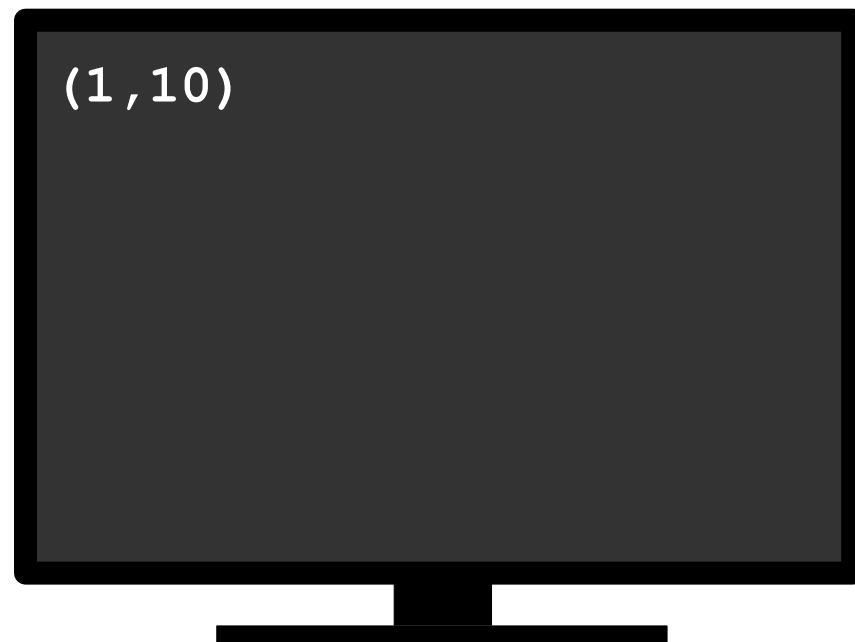
Imprimindo a lista

```
imprimeLista(phead);  
void imprimeLista(struct coord * p)  
{  
    if (p)  
    {  
        → imprimeElemento(p);  
        imprimeLista(p->next);  
    }  
    else  
        printf("Fim da Lista\n");  
}
```



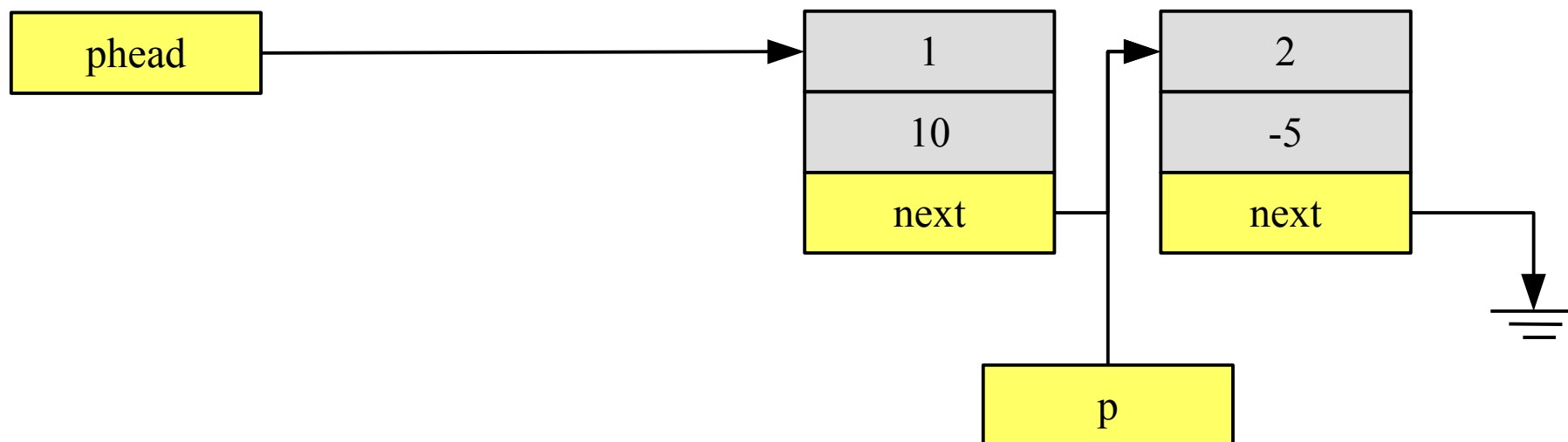
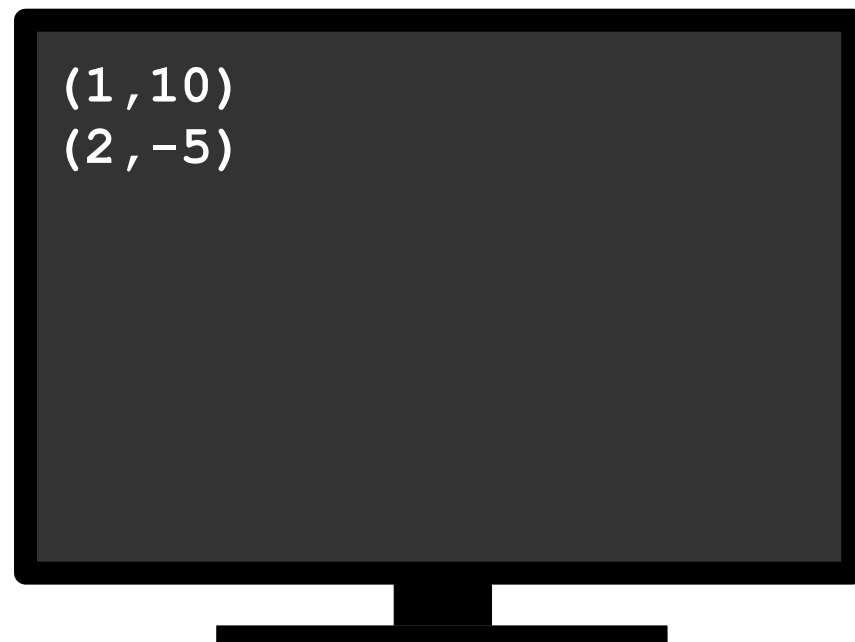
Imprimindo a lista

```
imprimeLista(phead);  
void imprimeLista(struct coord * p)  
{  
    if (p)  
    {  
        imprimeElemento(p);  
        → imprimeLista(p->next);  
    }  
    else  
        printf("Fim da Lista\n");  
}
```



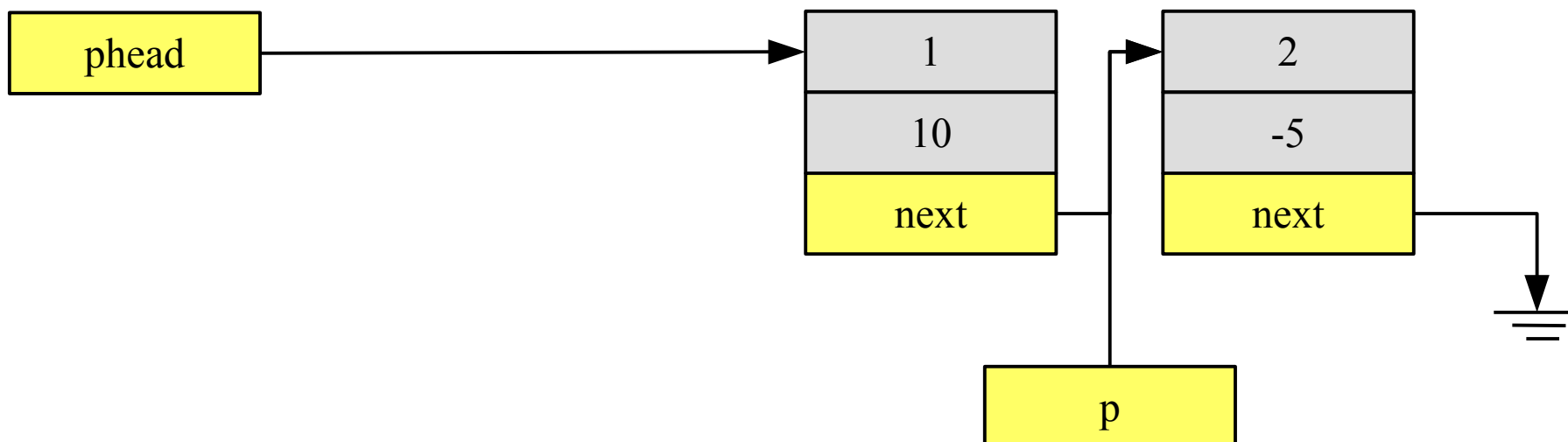
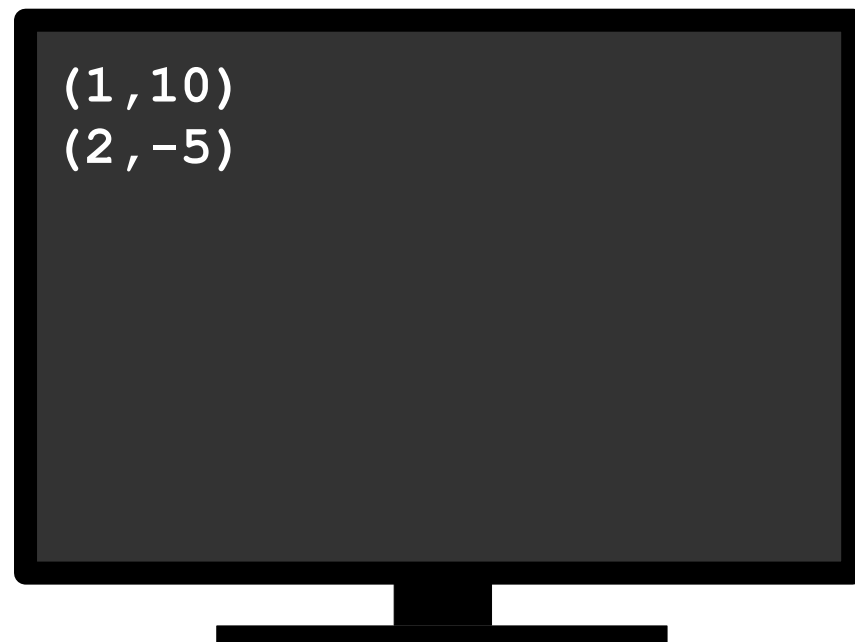
Imprimindo a lista

```
imprimeLista (phead) ;  
void imprimeLista (struct coord * p)  
{  
    if (p)  
    {  
        → imprimeElemento (p) ;  
        imprimeLista (p->next) ;  
    }  
    else  
        printf("Fim da Lista\n") ;  
}
```



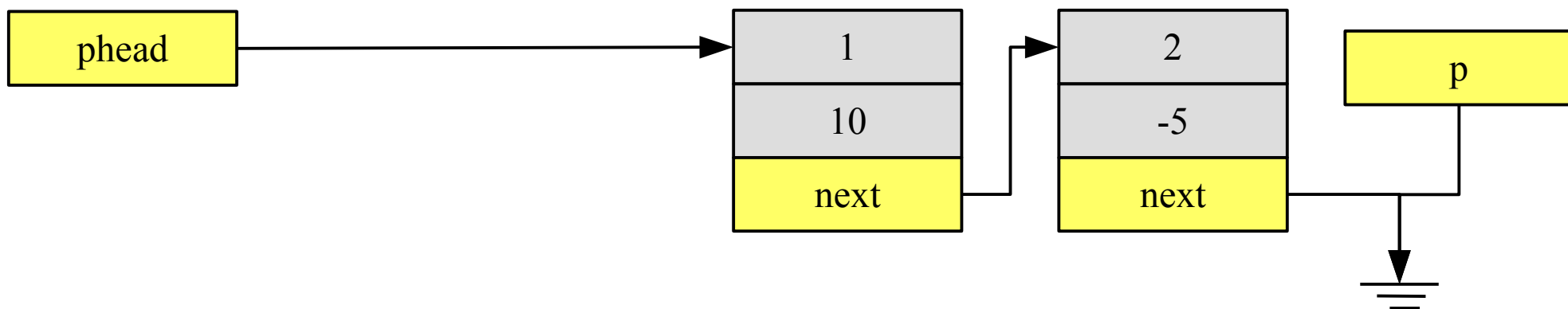
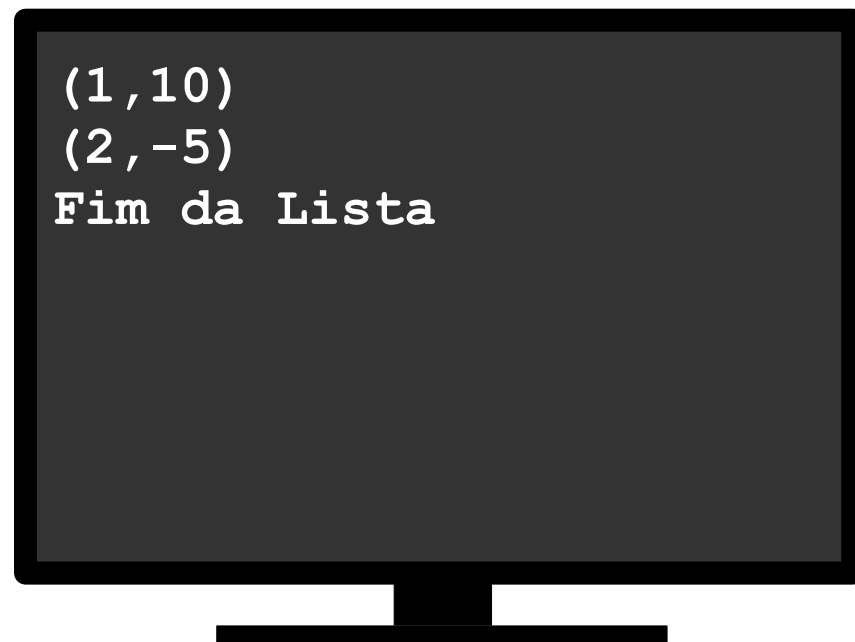
Imprimindo a lista

```
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");  
}
```

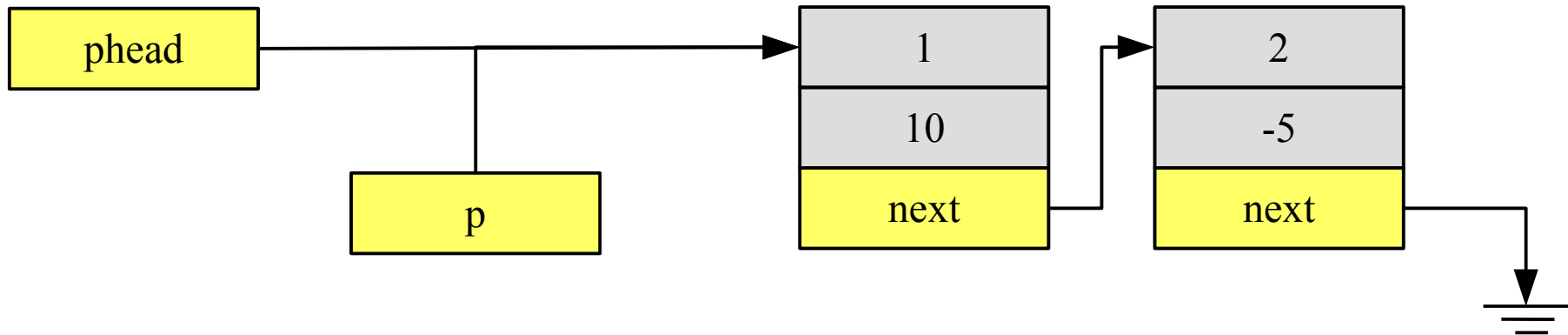


Buscando Elemento

```
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;  
}
```

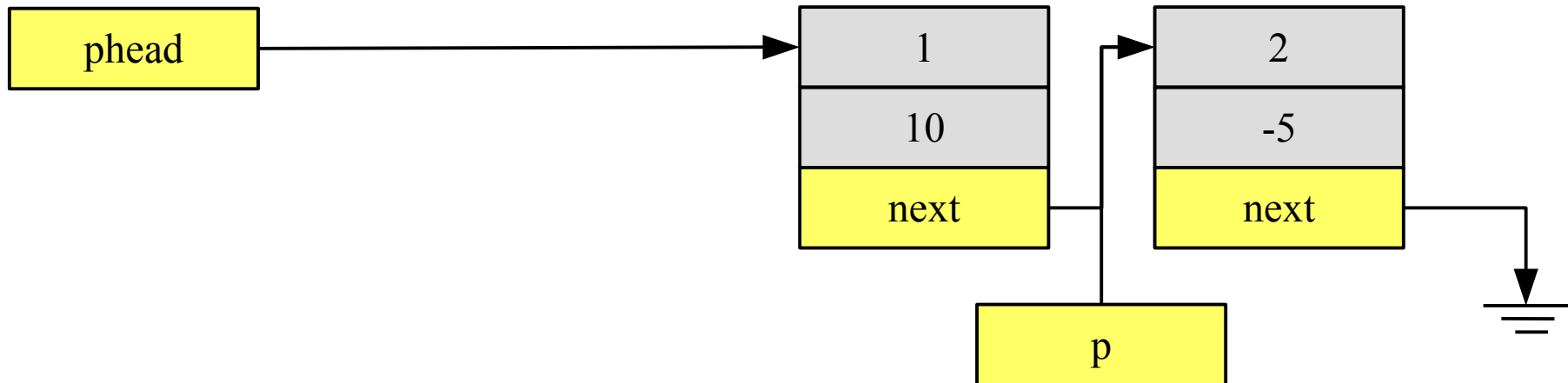
Buscando Elemento

```
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;  
}
```



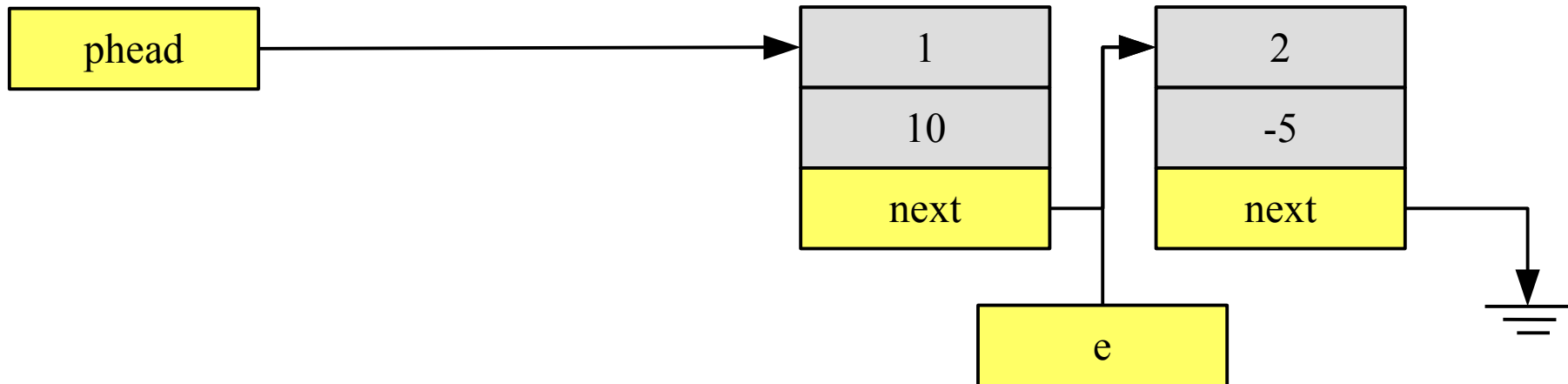
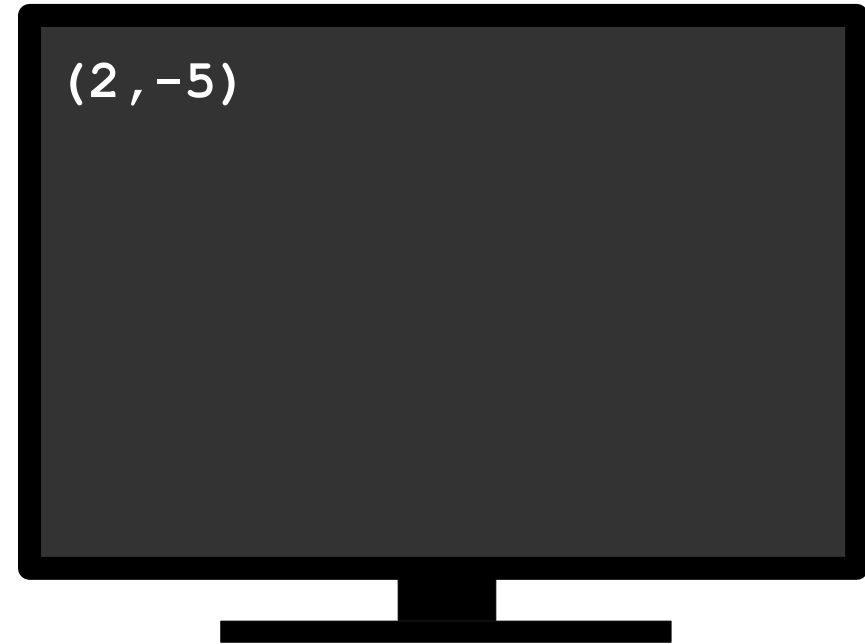
Buscando Elemento

```
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;  
}
```



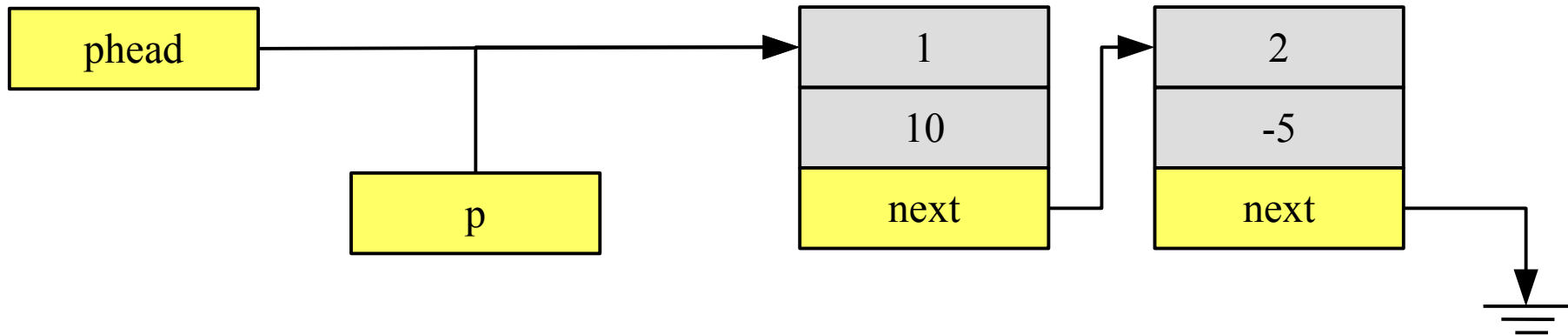
Buscando Elemento

```
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");  
}
```



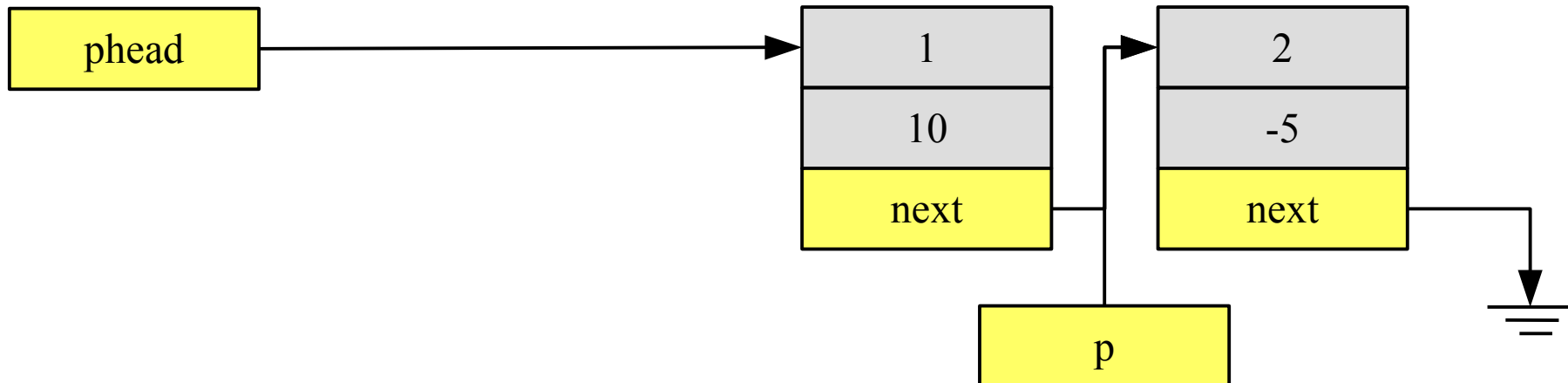
Buscando Elemento

```
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;  
}
```



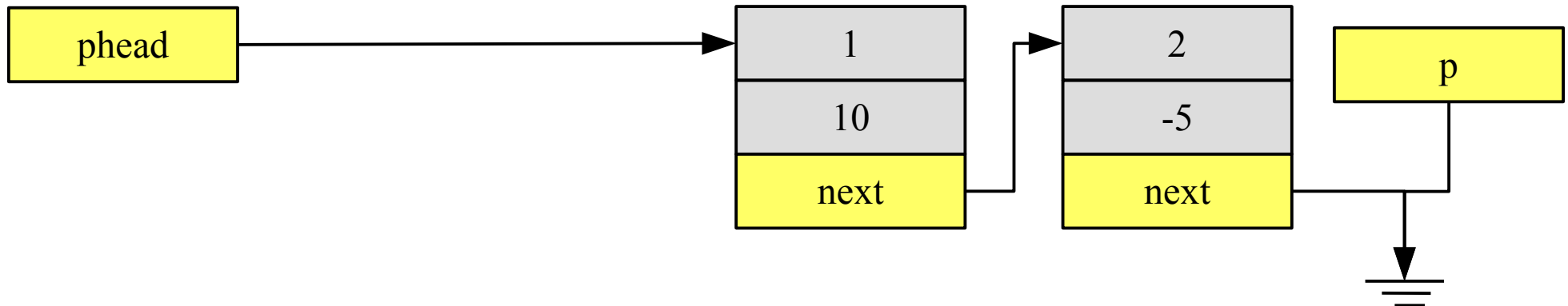
Buscando Elemento

```
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

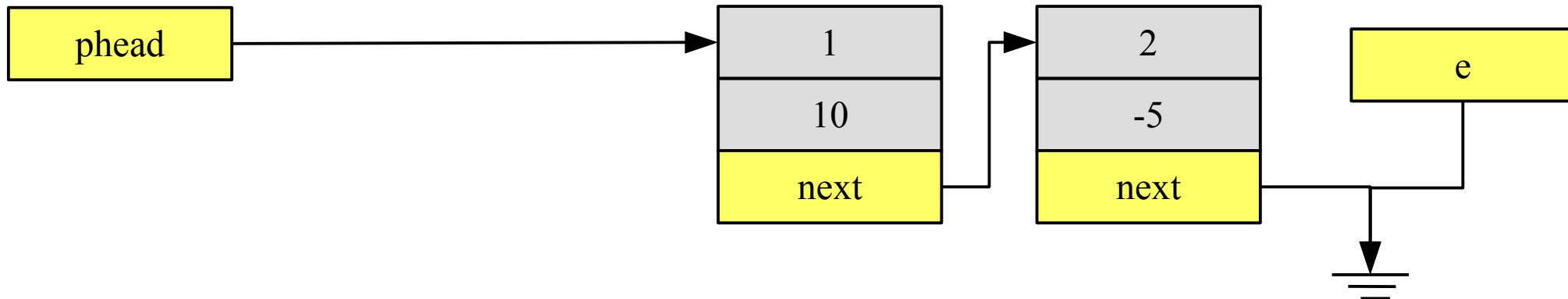
```
imprimeElemento(buscaCoord(3,-5,phead));  
struct coord * buscaCoord(int x, int y, struct coord * p)  
{  
    if (p)  
        if ((x==p->x) && (y==p->y))  
            return p;  
        else  
            return buscaCoord(x, y, p->next);  
    else  
        → return NULL;  
}
```



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");  
}
```

Elemento inexistente!



Exclusão em Listas Encadeadas

Excluindo Elemento (pelas coordenadas)

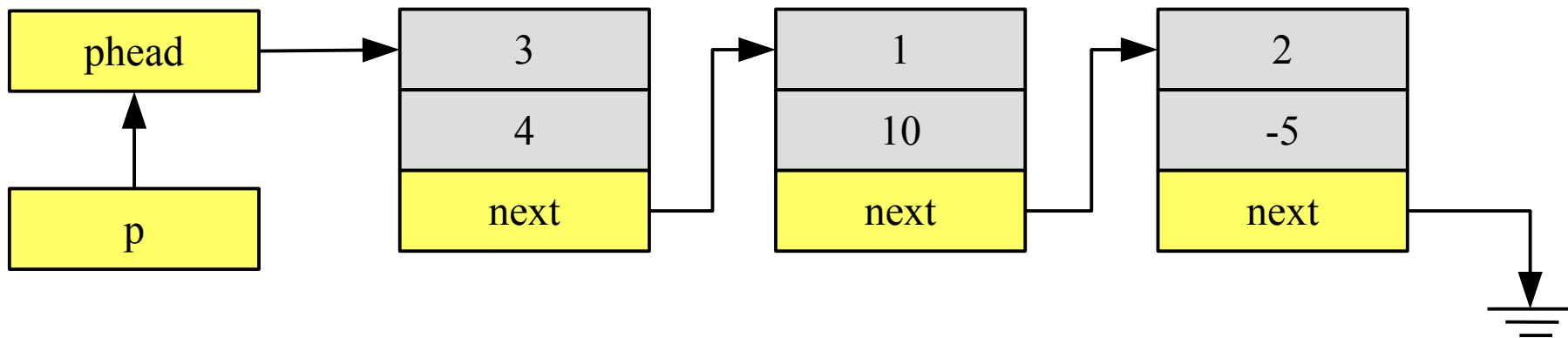
```
void excluiElementoCoord(int x, int y, struct coord ** p)
{
    if (*p)
    {
        if ((x==(*p)->x) && (y==(*p)->y))
        {
            struct coord * e = *p;
            *p = e->next;
            libera(e);
            printf("Elemento excluído com sucesso!\n");
        }
        else
   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

Liberando Memória

```
void libera(struct coord * e)
{
    free(e) ;
}
```

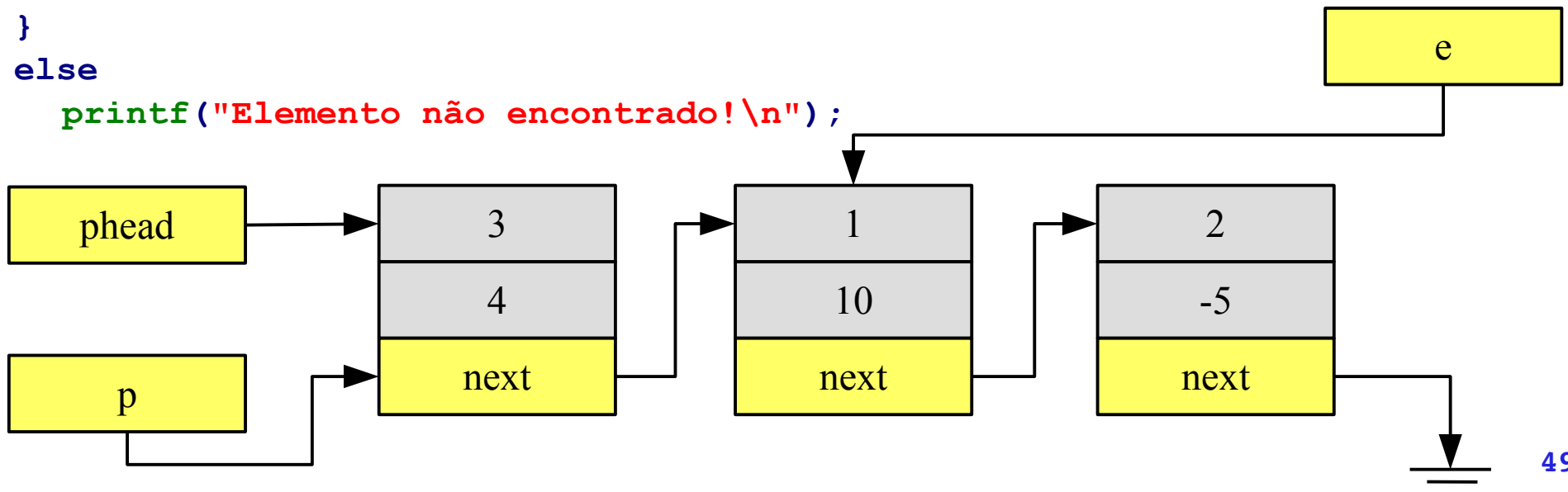
Excluindo Elemento (pelas coordenadas)

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



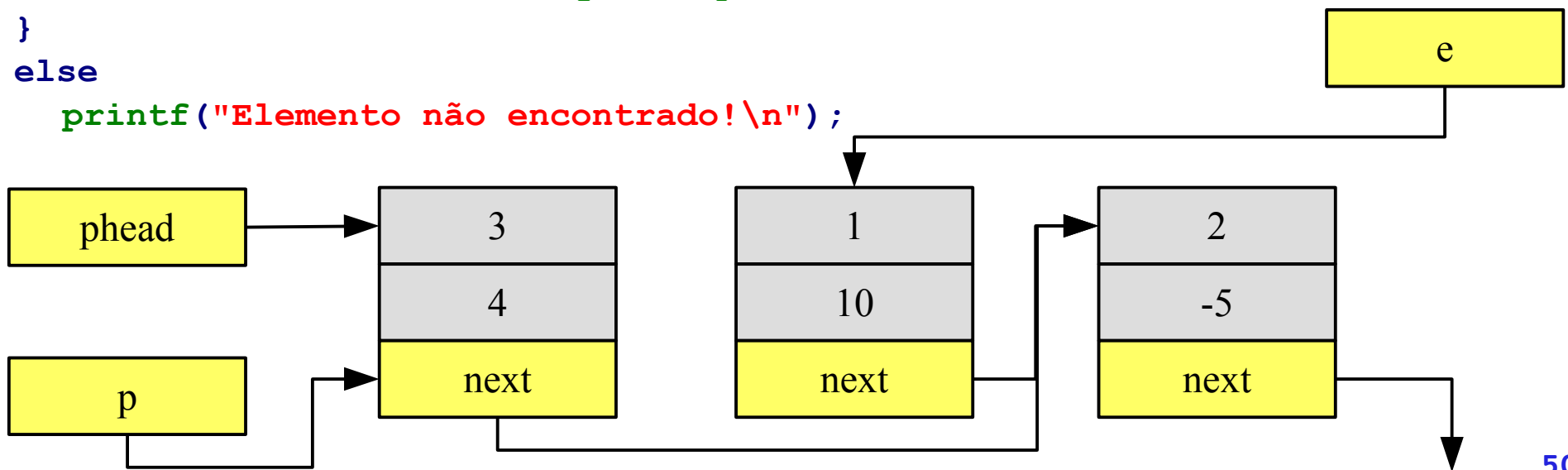
Excluindo Elemento (pelas coordenadas)

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



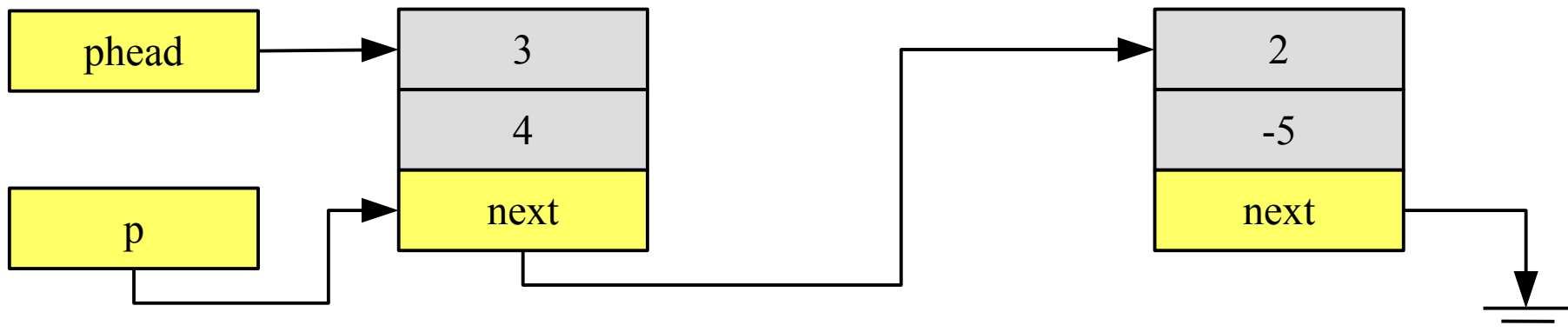
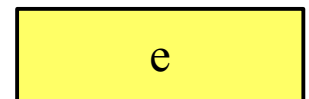
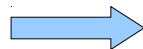
Excluindo Elemento (pelas coordenadas)

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



Excluindo Elemento (pelas coordenadas)

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



Excluindo Elemento (pelas coordenadas)

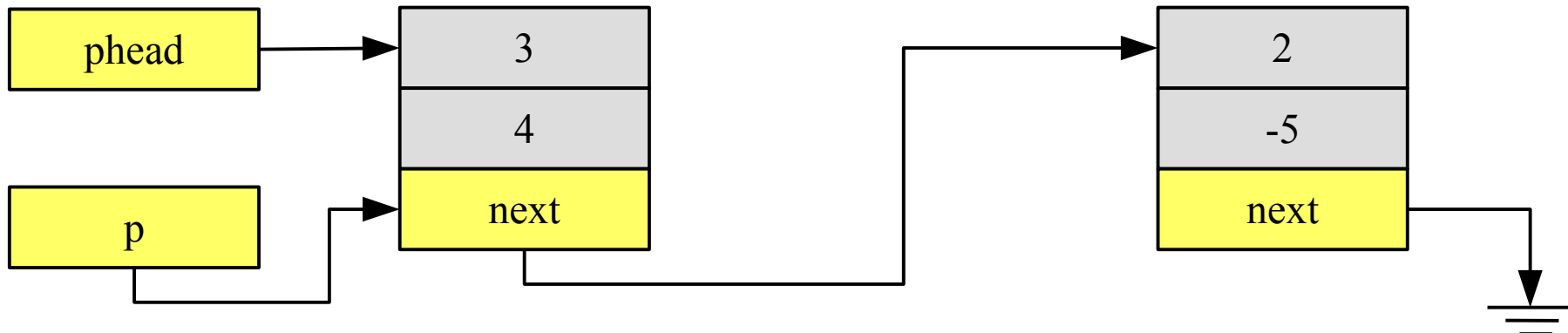
```
excluiElementoCoord(1,10,&phead);  
void excluiElementoCoord(int x, int y, struct coord ** p)  
{  
    if (*p)  
    {  
        if ((x==(*p)->x) && (y==(*p)->y))  
        {  
            struct coord * e = *p;  
            *p = e->next;  
            libera(e);  
            printf("Elemento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



Elemento excluído
com sucesso!



e



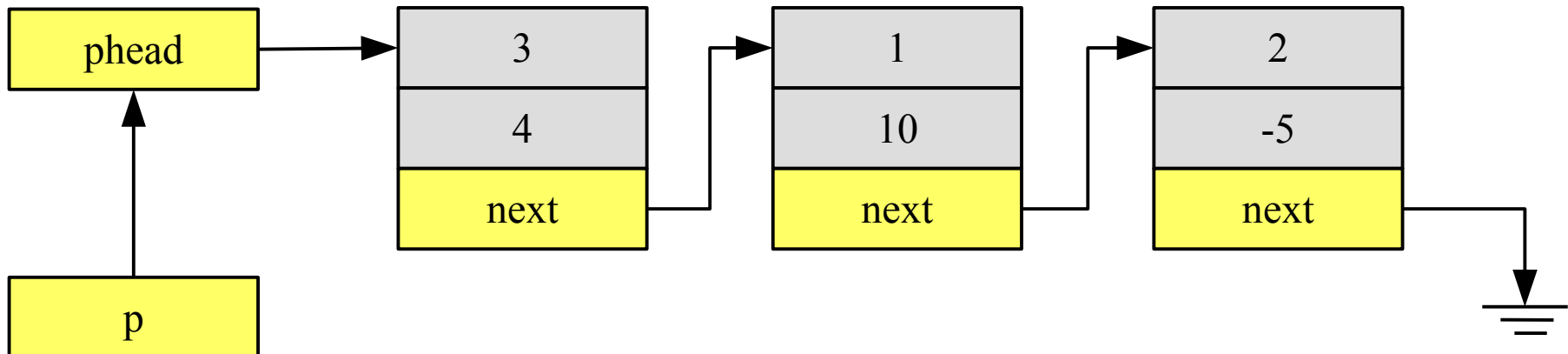
Excluindo Elemento (pelo apontador)

```
void excluiElementoPonteiro(struct coord * e, struct coord
** p)
{
    if (*p)
    {
        if (*p == e)
        {
            *p = e->next;
            libera(e);
            printf("Elemento excluído com sucesso!\n");
        }
        else
            excluiElementoPonteiro(e, &((*p)->next));
    }
    else
        printf("Elemento não encontrado!\n");
}
```

Excluindo Toda a Lista

```
limparLista(&phead)
```

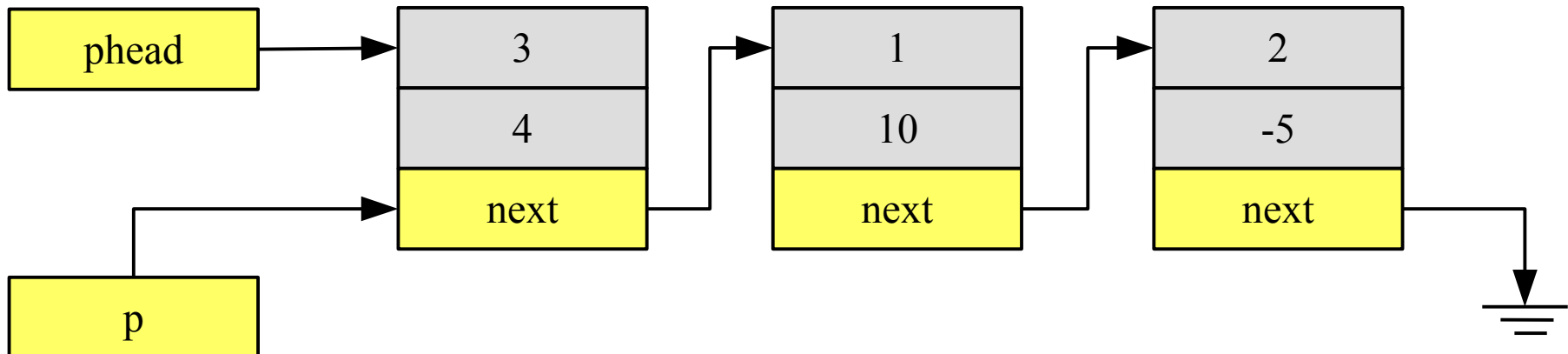
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

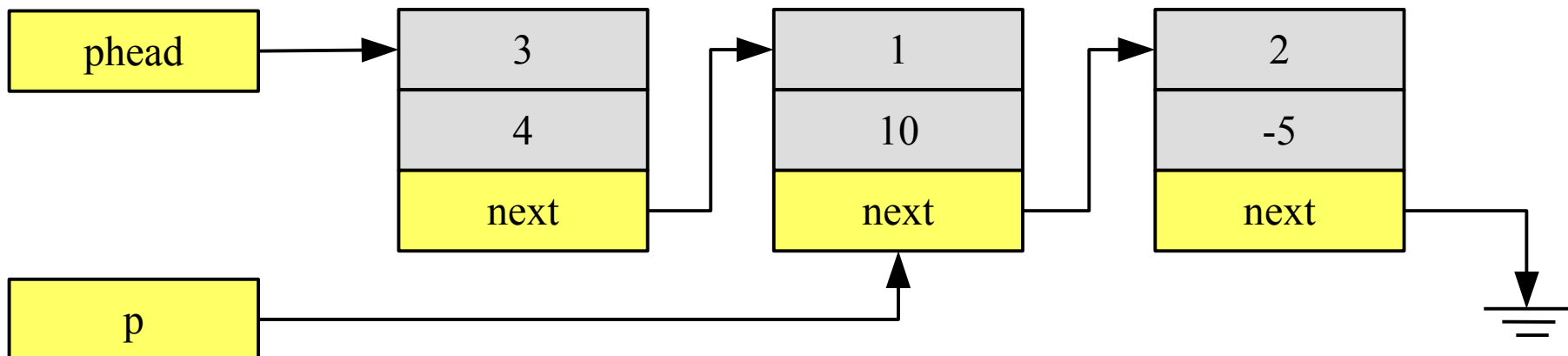
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

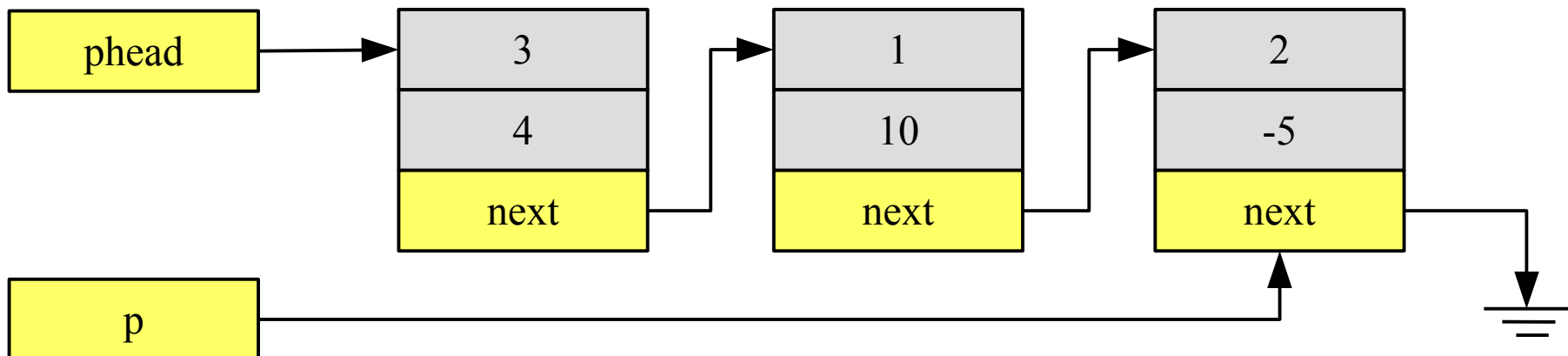

```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        → limparLista(&((*p)->next))
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

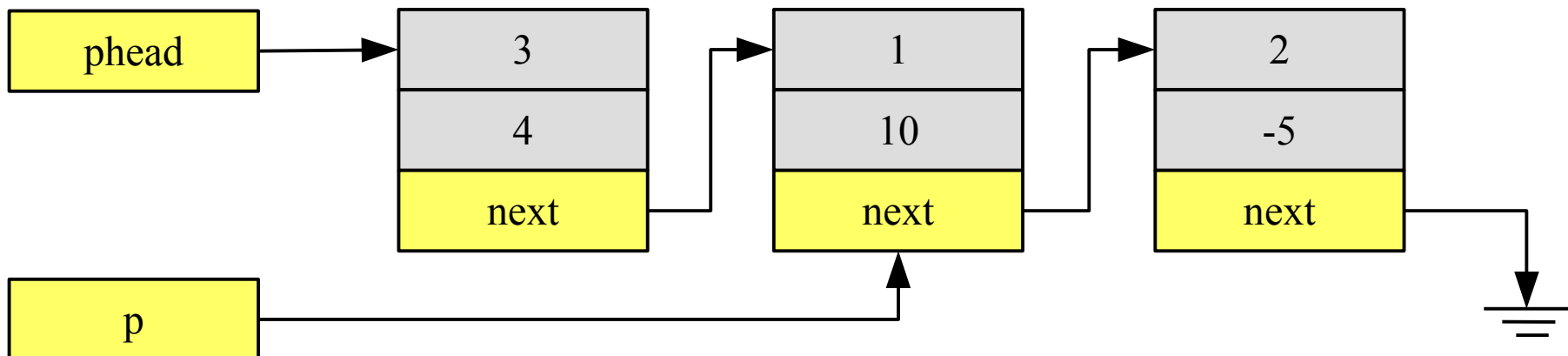
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

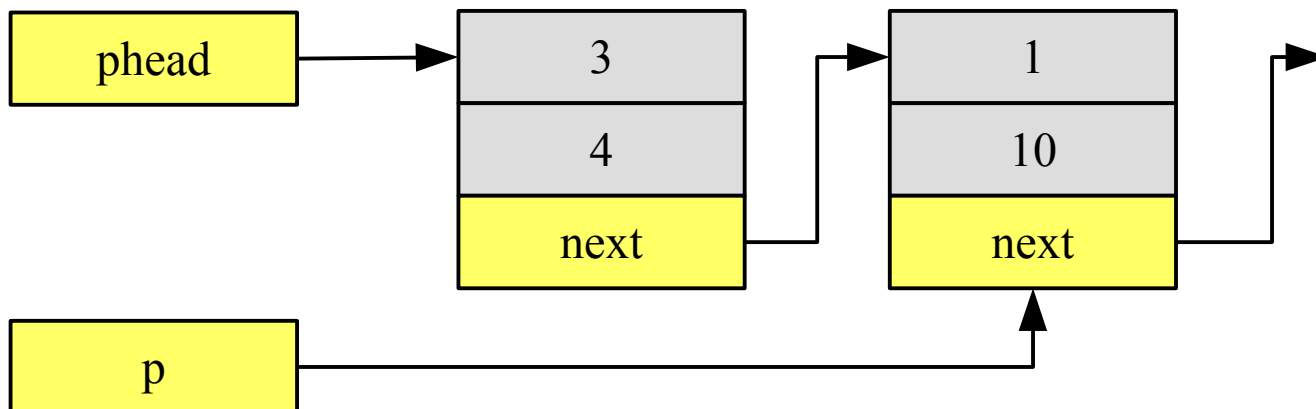
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

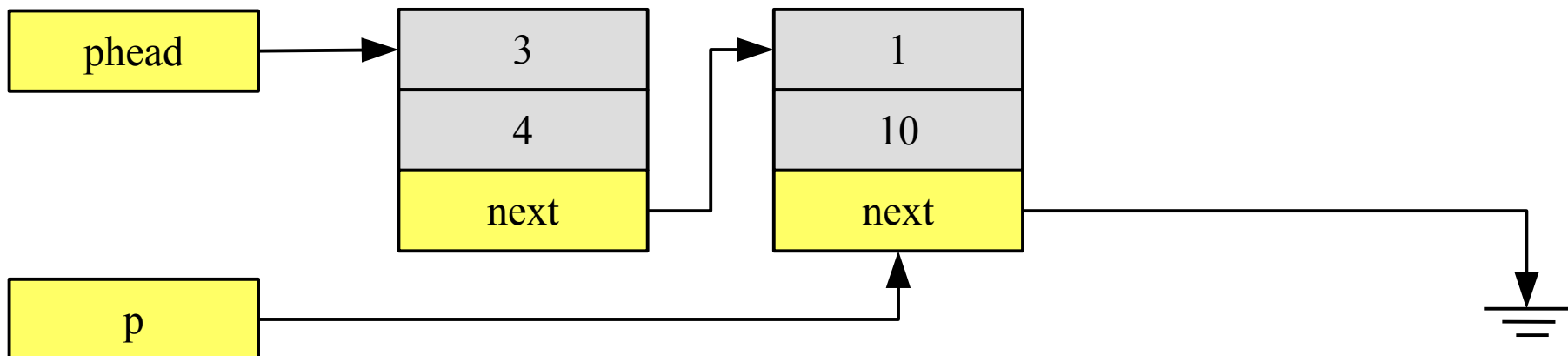
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

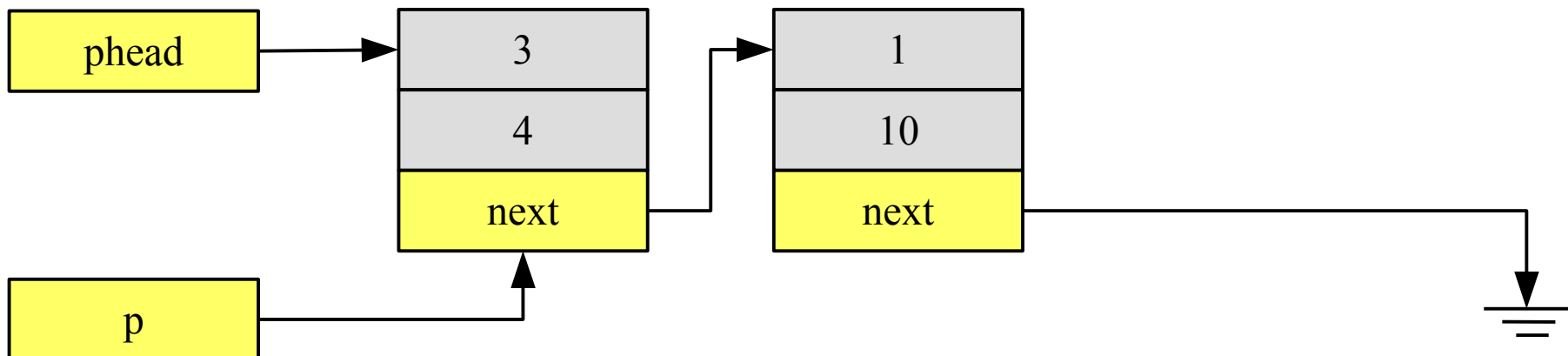
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        libera(*p);
        → *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

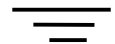
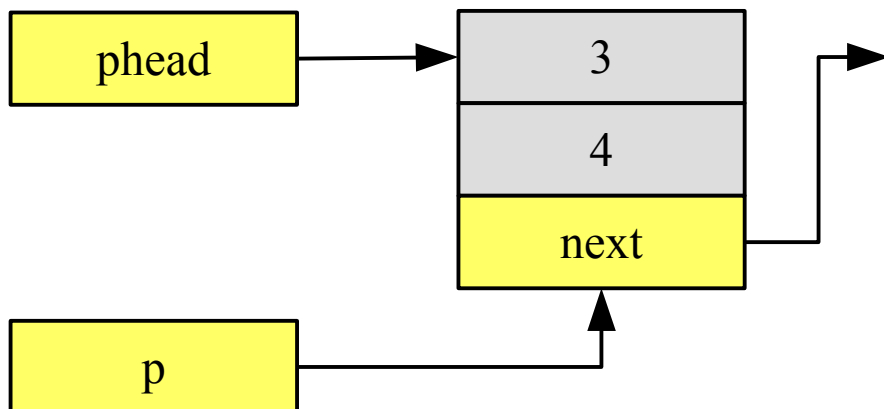
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

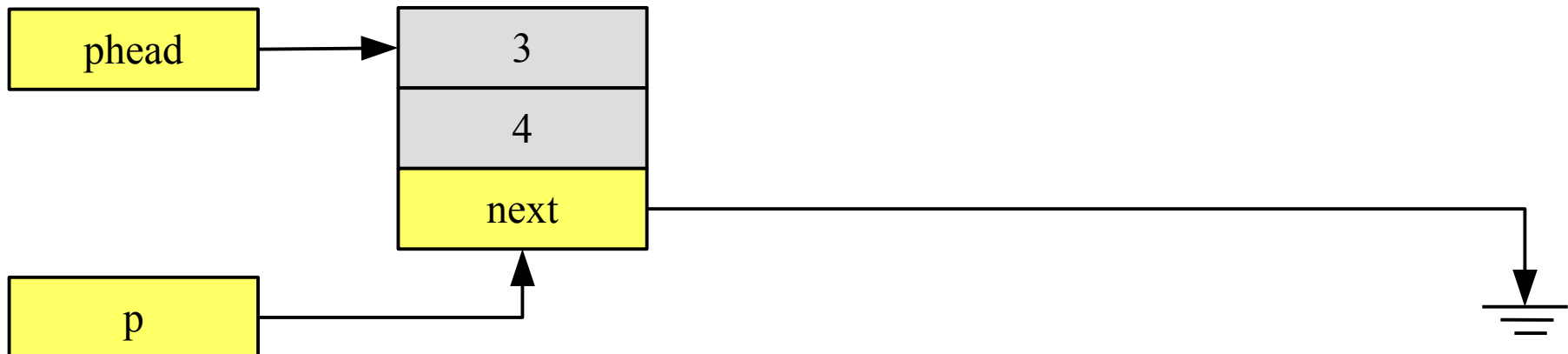
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

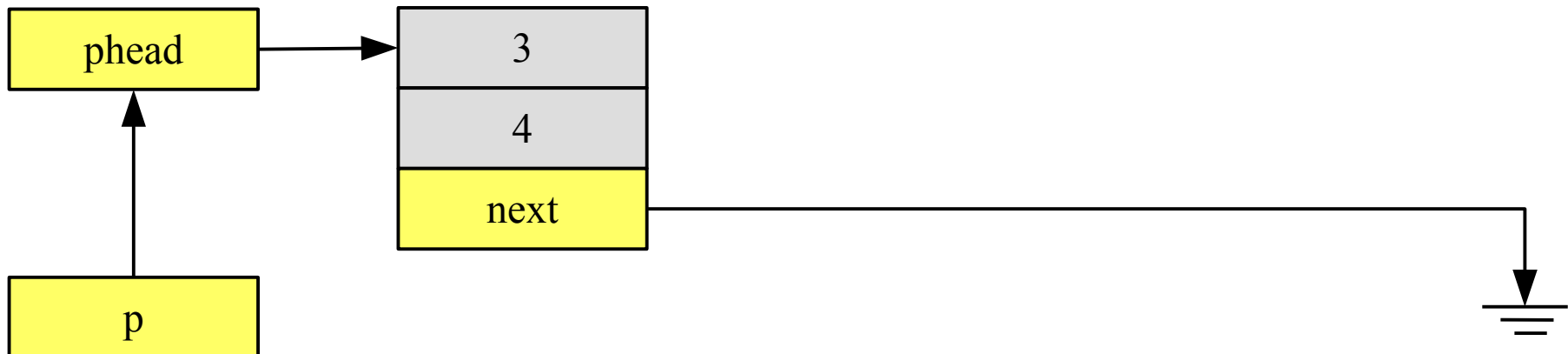
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

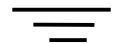
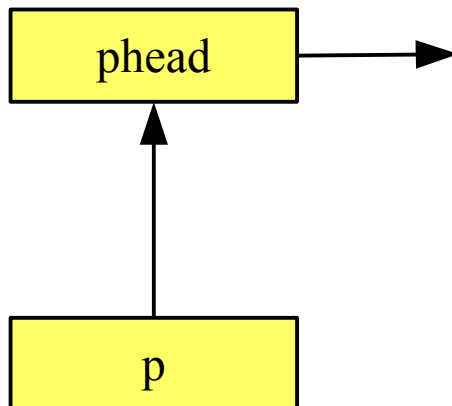
```
void limparLista(struct coord ** p)
{
    if (*p)
    {
        limparLista(&((*p)->next))
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Lista

```
limparLista(&phead)
```

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



Excluindo Toda a Lista

```
limparLista(&phead)
```

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



Usando isso tudo!

```
int main()
{
    struct coord * phead = NULL;
    int i;
    //Inserir 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);

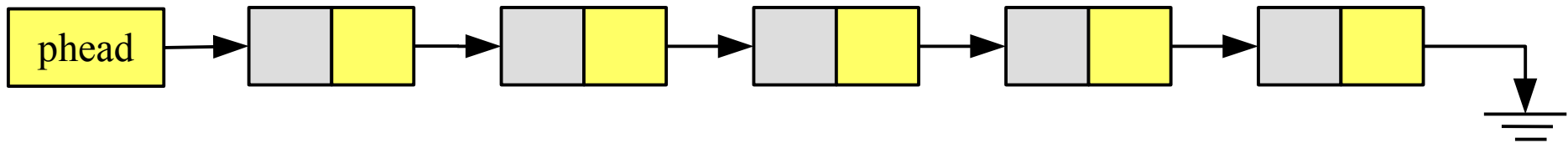
excluiElementoPonteiro(buscaCoord(2, 20, phead),
                        &phead);
imprimeLista(phead);

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

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

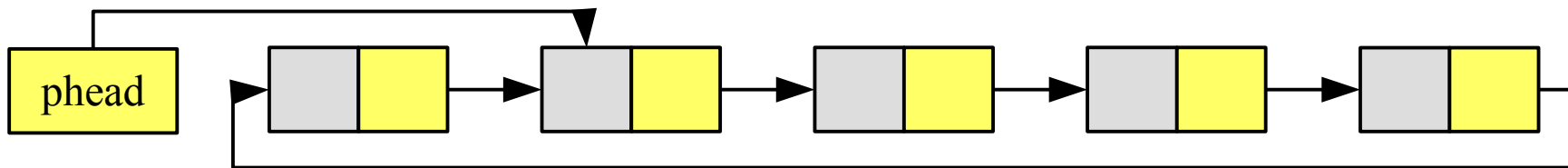
Alguns tipos de lista encadeada

- Lista encadeada simples.



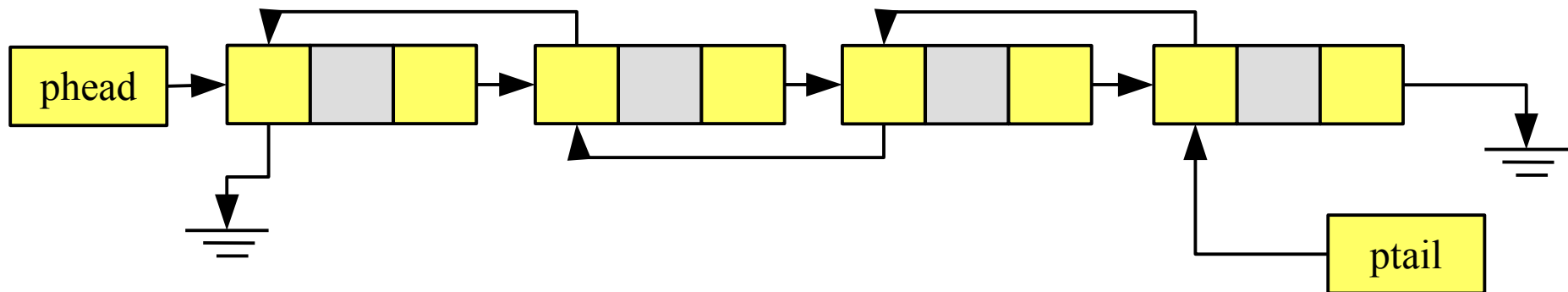
- Lista encadeada circular

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

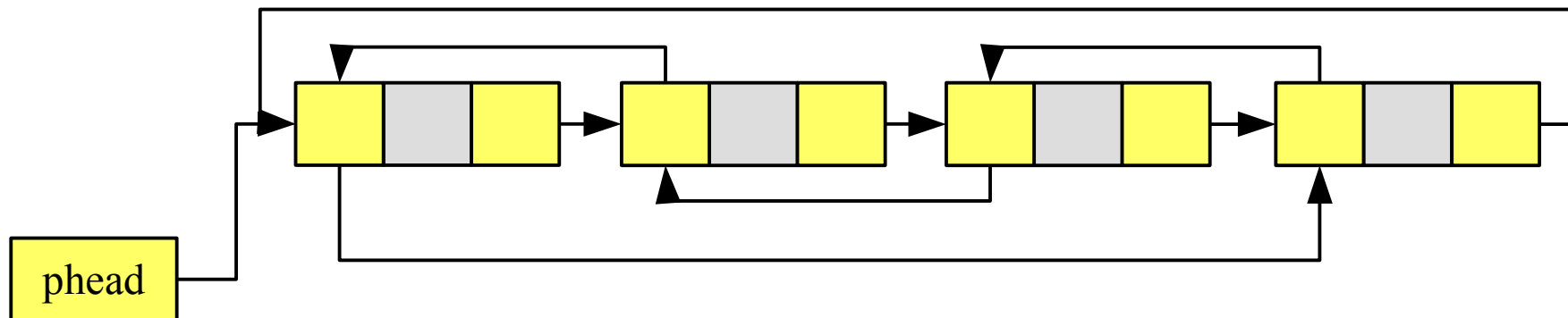


Alguns tipos de lista encadeada

- Lista duplamente encadeada com sentinelas.



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



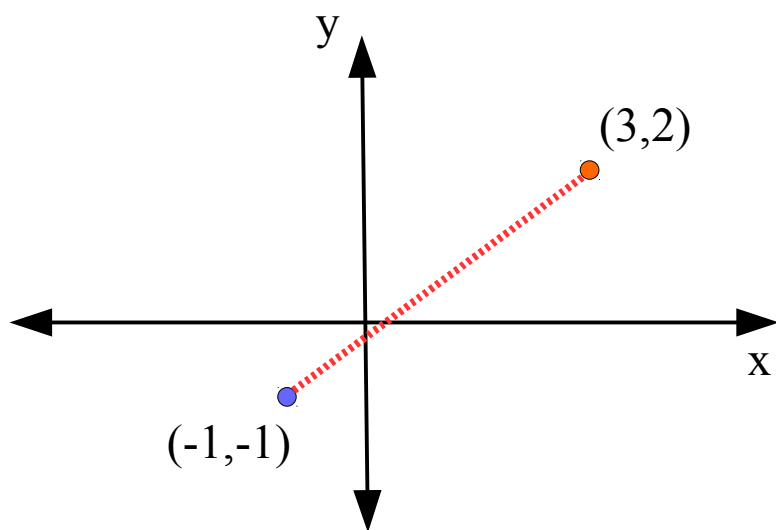
Exercícios

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.

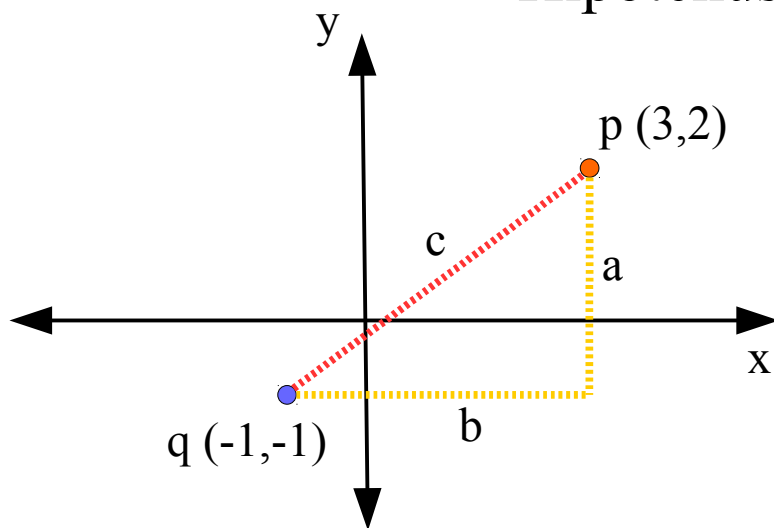
Exercício

- 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:

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

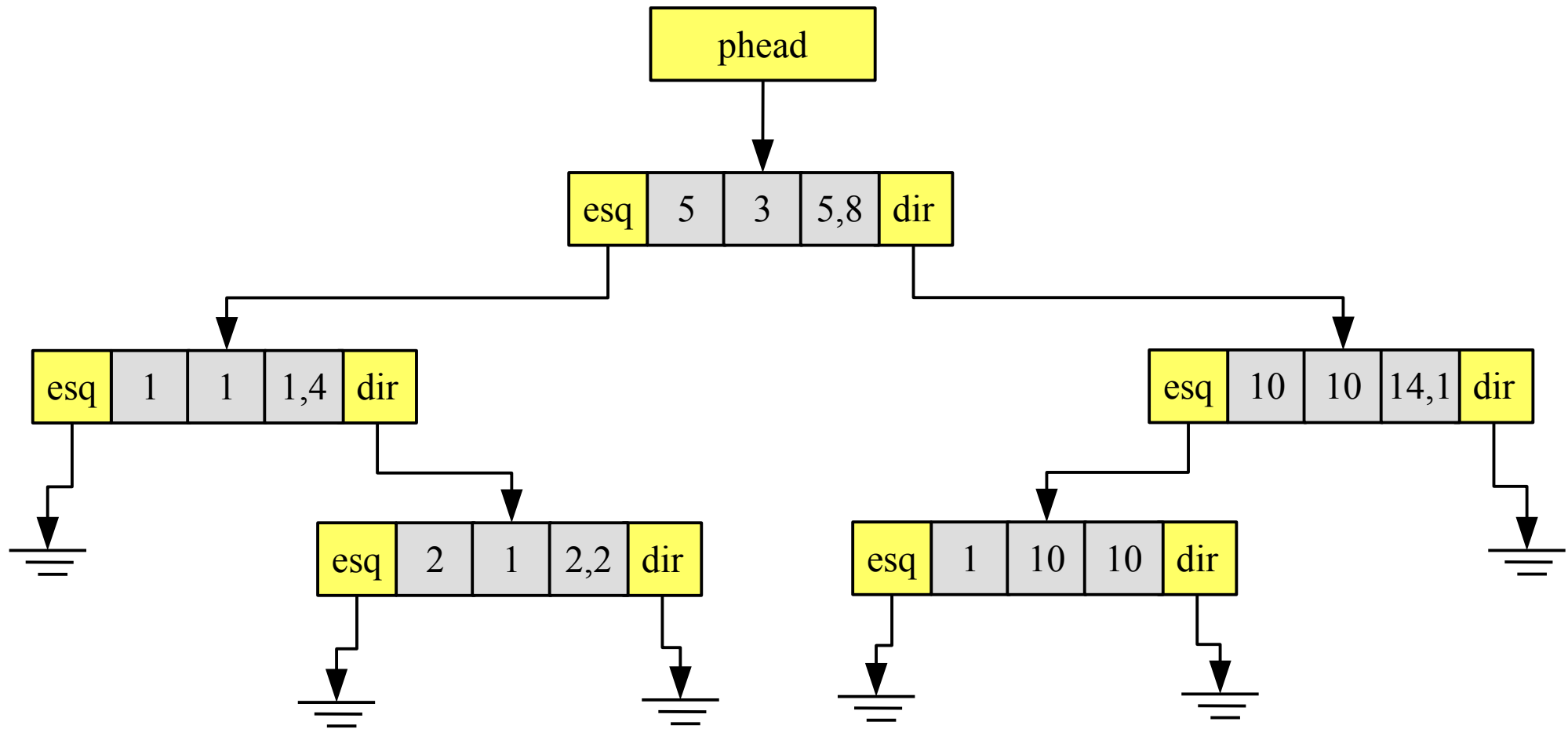
Árvores Binárias

O que é uma árvore binária?

- É uma estrutura de dados onde:
 - Os elementos estão espalhados em qualquer lugar da memória (em qualquer ordem).
 - Temos um apontador para o primeiro elemento da árvore
 - Cada elemento da árvore aponta para dois filhos, o esquerdo e o direito.
 - Os filhos com valor menor ficam do lado esquerdo e os maiores no lado direito

O que é uma árvore binária?

- Exemplo de árvore binária de coordenadas no R^2 ordenada pela distância euclidiana em relação à origem (0,0):



Declaração de uma árvore binária vazia

```
struct coord
{
    short x,y;
    float d;
    struct coord * esq;
    struct coord * dir;
};

struct coord * phead = NULL;
```

Criando um elemento

```
struct coord * criaElem( short x, short y)
{
    struct coord * p = (struct coord *)
        malloc(sizeof(struct coord)) ;
    p->x = x;
    p->y = y;
    p->d = distEuclid(p->x, p->y) ;
    p->esq = NULL;
    p->dir = NULL;
    return p;
}
```

Inserção em Árvores Binárias

Inserindo um elemento sem balancear a árvore

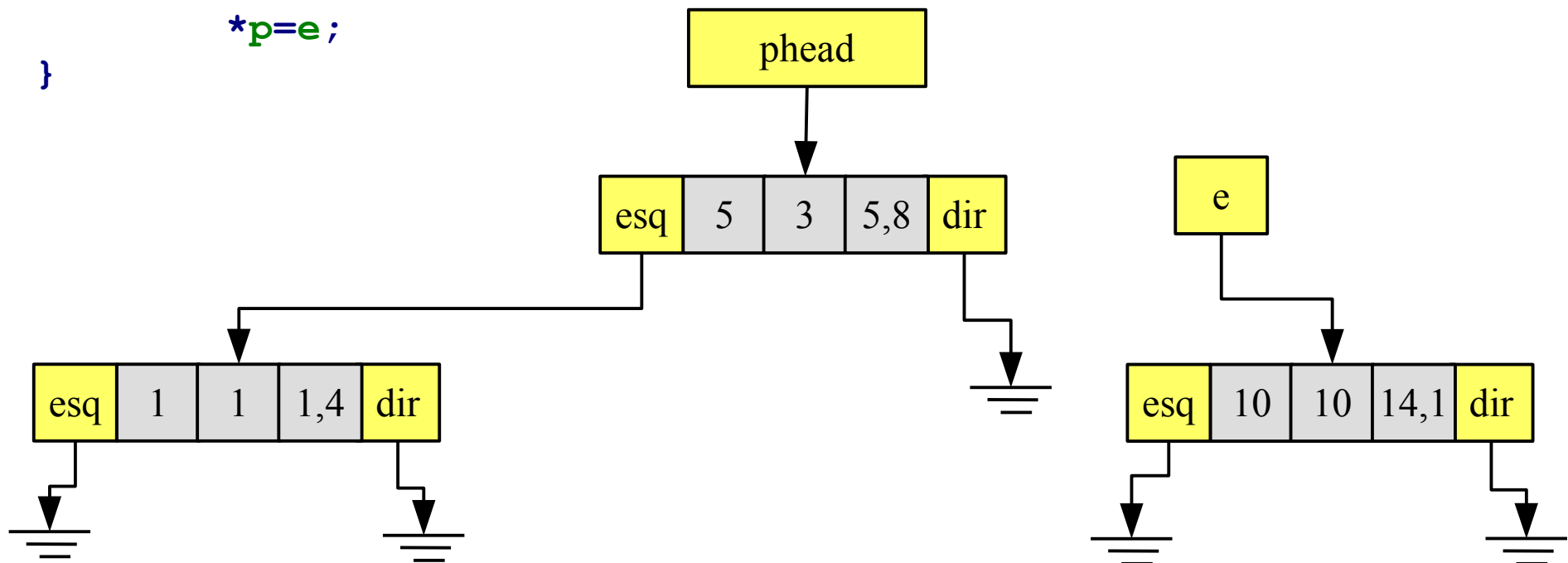
```
void insereDesbalanceado(struct coord * e, struct
coord ** p)
{
    if (*p)
    {
        if ( e->d > (*p)->d )
            insereDesbalanceado(e, &((*p)->dir));
        else
            insereDesbalanceado(e, &((*p)->esq));
    }
    else
        *p=e;
}
```

Inserindo um elemento sem balancear a árvore

```
insereDesbalanceado (e , &phead) ;
```

```
void insereDesbalanceado(struct coord * e, struct coord ** p)
{
    if (*p)
    {
        if ( e->d > (*p)->d )
            insereDesbalanceado(e, &((*p)->dir));
        else
            insereDesbalanceado(e, &((*p)->esq));
    }
    else
        *p=e;
}
```

A yellow rectangular box with a black border contains the text "phead". An arrow originates from the box and points to the variable "p" in the function call "insereDesbalanceado(p, e)" within the code snippet above.

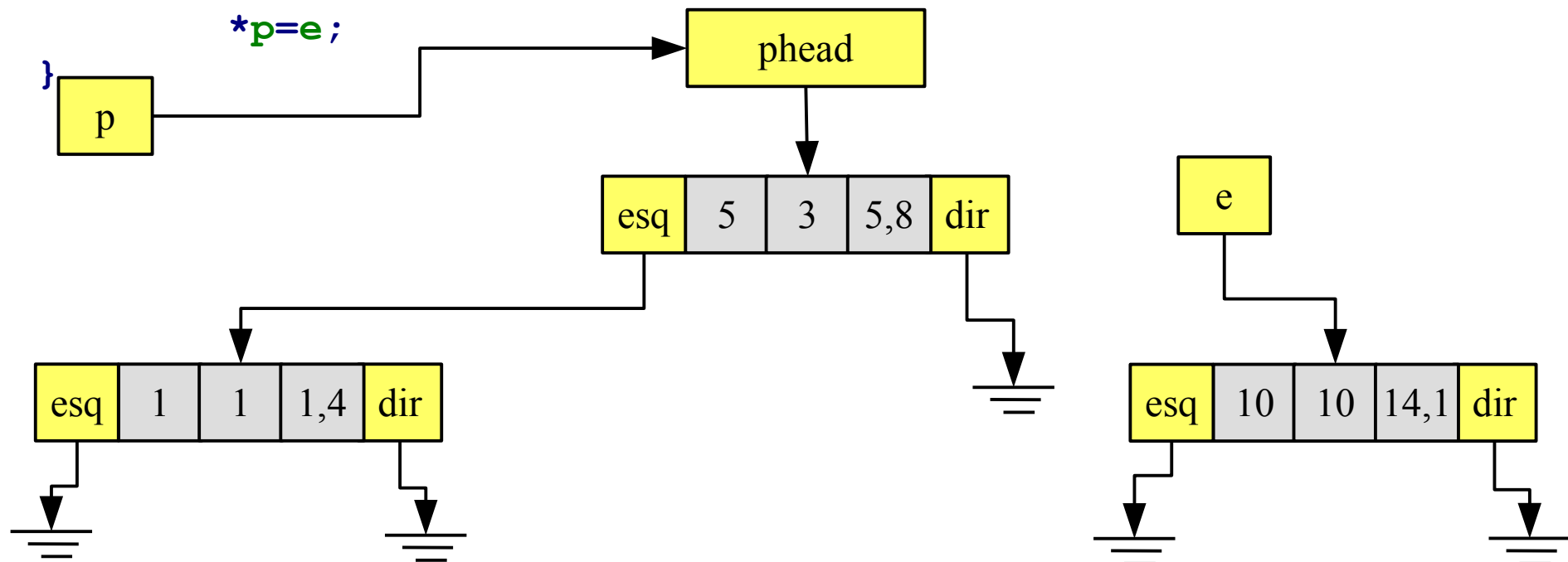


Inserindo um elemento sem balancear a árvore

```
insereDesbalanceado(e , &phead) ;
```

```
void insereDesbalanceado(struct coord * e, struct coord ** p)
```

```
{  
    if (*p)  
    {  
        if ( e->d > (*p)->d )  
            insereDesbalanceado(e, &((*p)->dir)) ;  
        else  
            insereDesbalanceado(e, &((*p)->esq)) ;  
    }  
    else  
        *p=e ;  
}
```



Inserindo um elemento sem balancear a árvore

```
insereDesbalanceado(e , &phead) ;
```

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

```
    if (*p)
```

```
    {
```

```
        → if ( e->d > (*p)->d )
```

```
            insereDesbalanceado(e, &((*p)->dir)) ;
```

```
        else
```

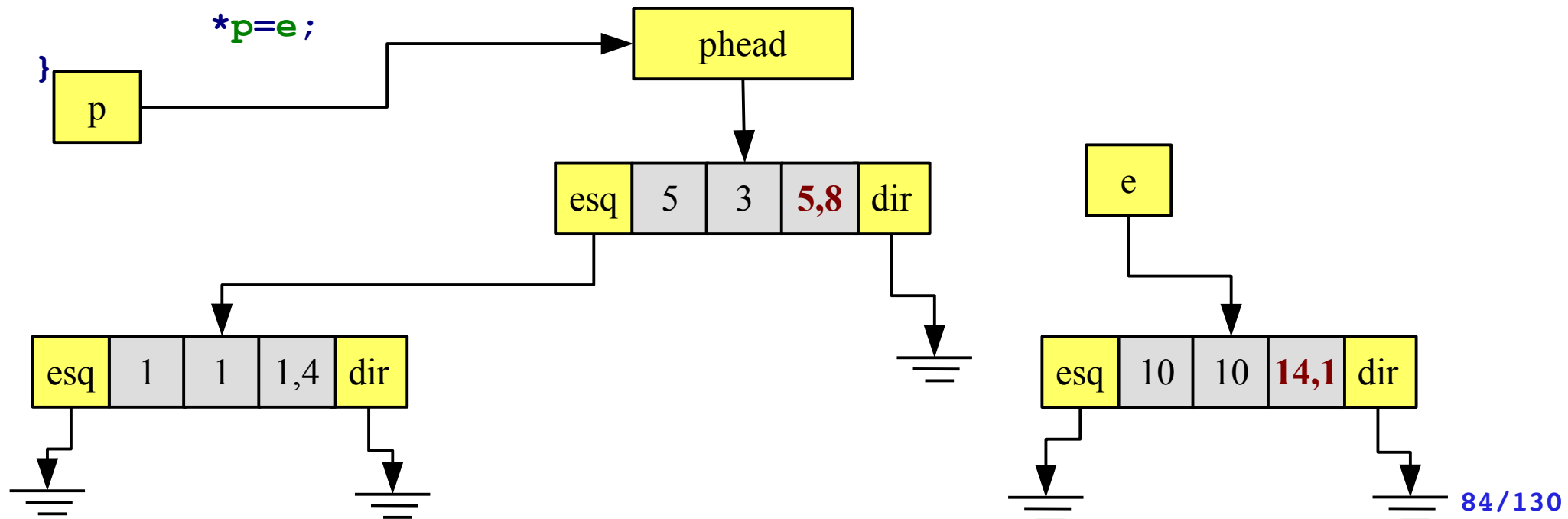
```
            insereDesbalanceado(e, &((*p)->esq)) ;
```

```
    }
```

```
    else
```

```
        *p=e;
```

```
}
```



Inserindo um elemento sem balancear a árvore

```
insereDesbalanceado(e , &phead) ;
```

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

```
    if (*p)
```

```
    {
```

```
        if ( e->d > (*p)->d )
```



```
        insereDesbalanceado(e, &((*p)->dir)) ;
```

```
    else
```

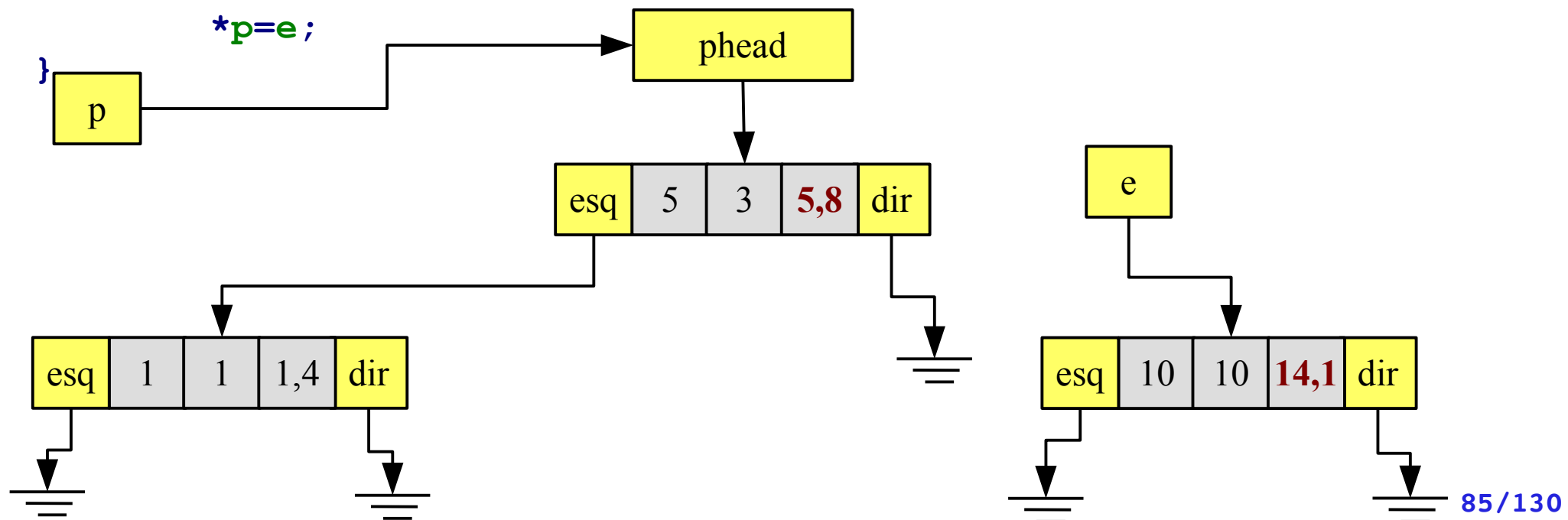
```
        insereDesbalanceado(e, &((*p)->esq)) ;
```

```
    }
```

```
    else
```

```
        *p=e;
```

```
}
```

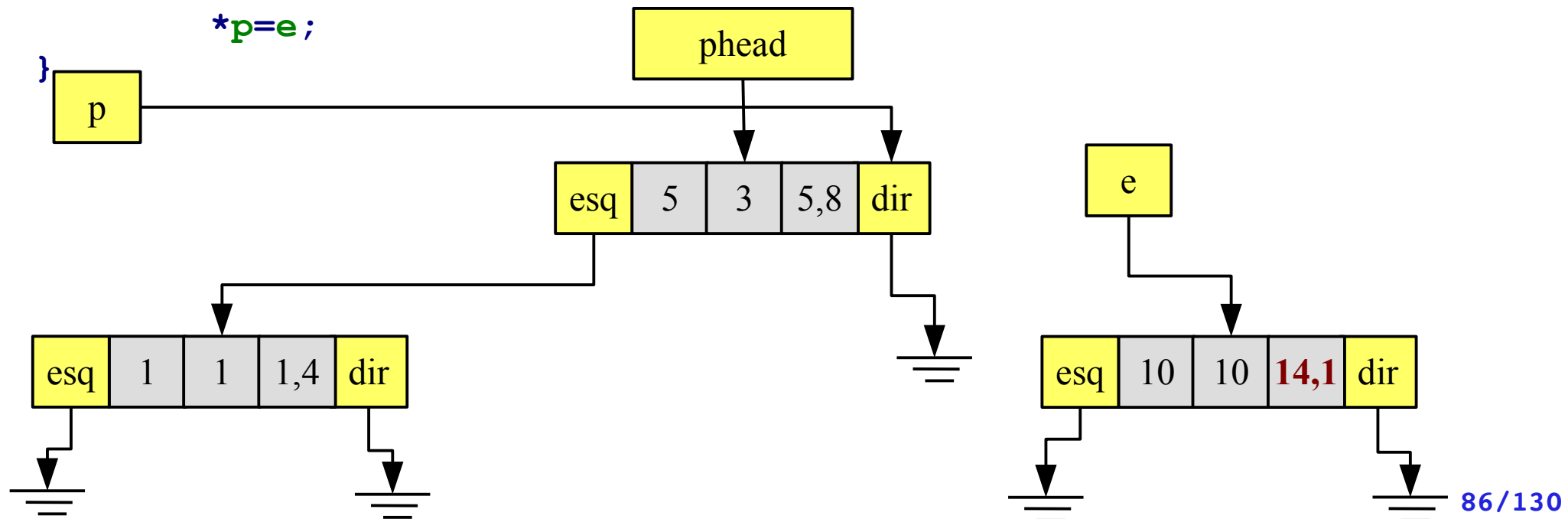


Inserindo um elemento sem balancear a árvore

```
insereDesbalanceado(e , &phead) ;
```

```
void insereDesbalanceado(struct coord * e, struct coord ** p)
```

```
{  
    if (*p)  
    {  
        if ( e->d > (*p)->d )  
            insereDesbalanceado(e, &((*p)->dir)) ;  
        else  
            insereDesbalanceado(e, &((*p)->esq)) ;  
    }  
    else  
        *p=e ;  
}
```



Inserindo um elemento sem balancear a árvore

```
insereDesbalanceado (e , &phead) ;
```

```
void insereDesbalanceado(struct coord * e, struct coord ** p)
```

{

```
if (*p)
```

{

```
if ( e->d > (*p)->d )
```

```
insereDesbalanceado(e, &((*p)->dir));
```

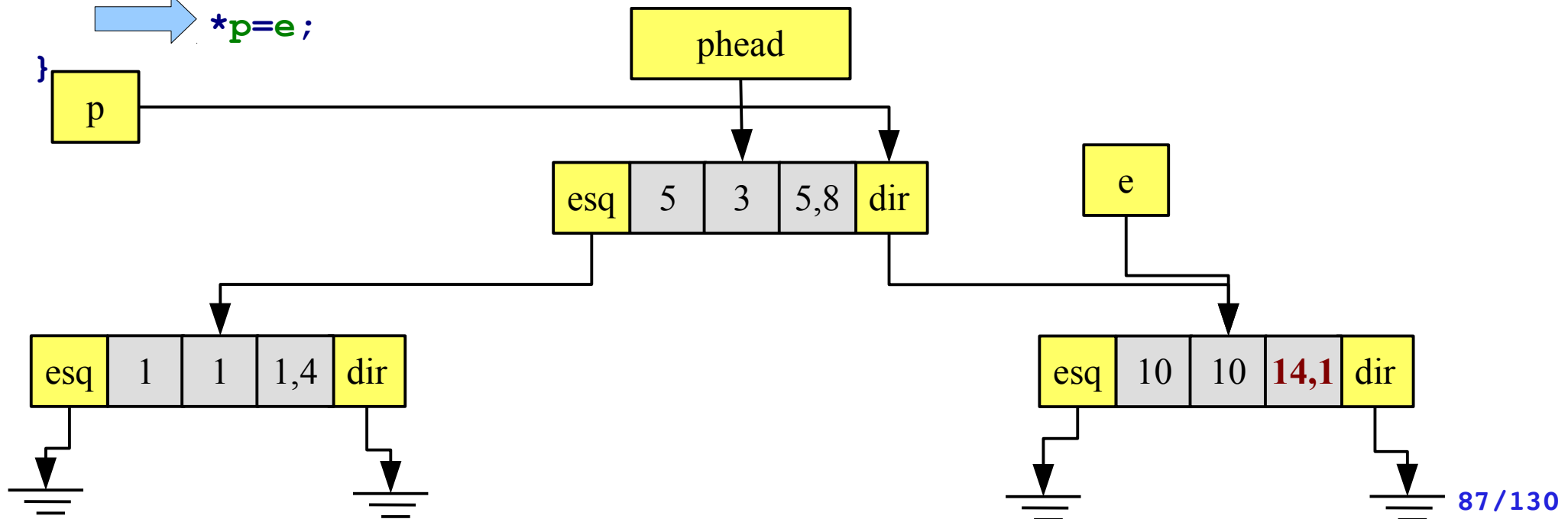
else

```
insereDesbalanceado(e, &((*p)->esq));
```

}

else

```
*p=e;
```



Percorrendo Árvores Binárias

Imprimindo um elemento

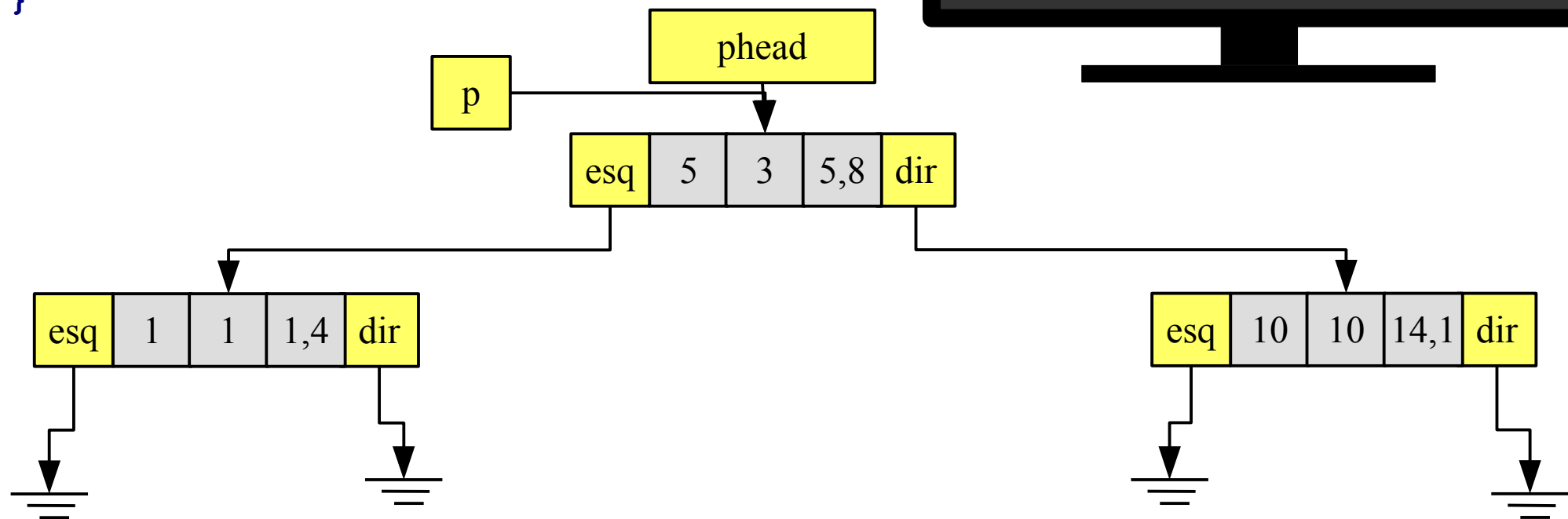
```
void imprimeElemento(struct coord * e)
{
    if (e)
        printf("( %hd, %hd) %f\n", e->x, e->y, e->d) ;
    else
        printf("Elemento inexistente!\n") ;
}
```

Imprimindo a árvore

```
void imprimeArvoreBin (struct coord * p)
{
    if (p)
    {
        imprimeArvoreBin (p->esq) ;
        imprimeElemento (p) ;
        imprimeArvoreBin (p->dir) ;
    }
}
```

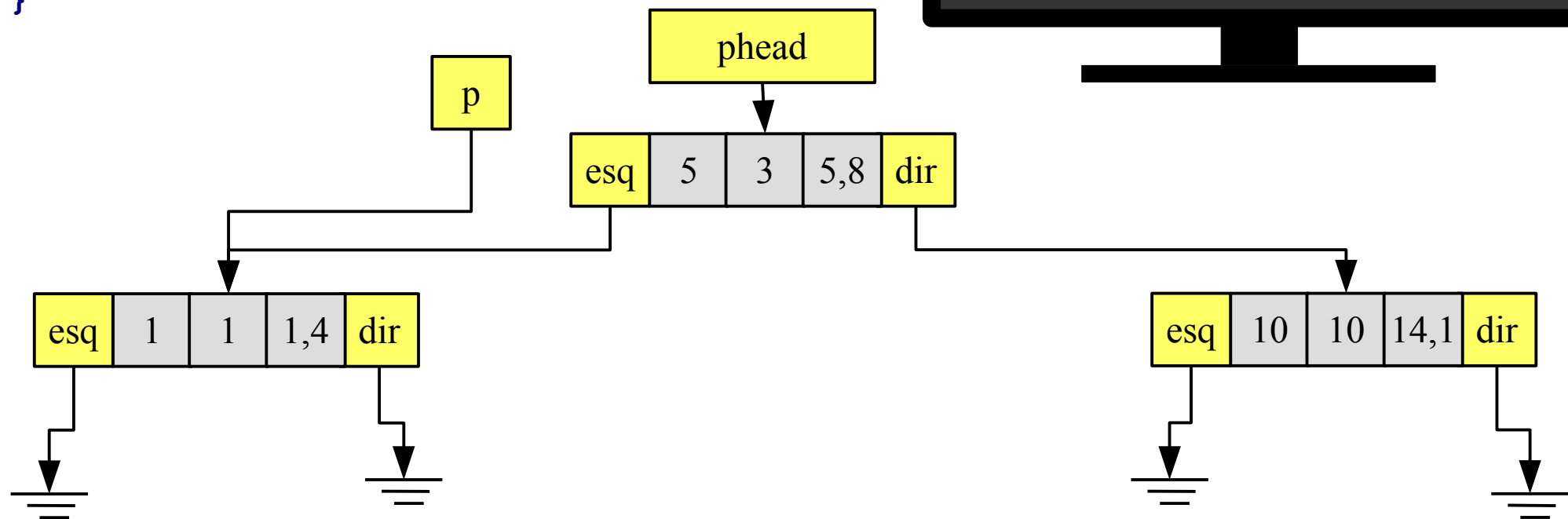
Imprimindo a árvore

```
imprimeArvoreBin(phead);  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        → imprimeArvoreBin(p->esq);  
        imprimeElemento(p);  
        imprimeArvoreBin(p->dir);  
    }  
}
```



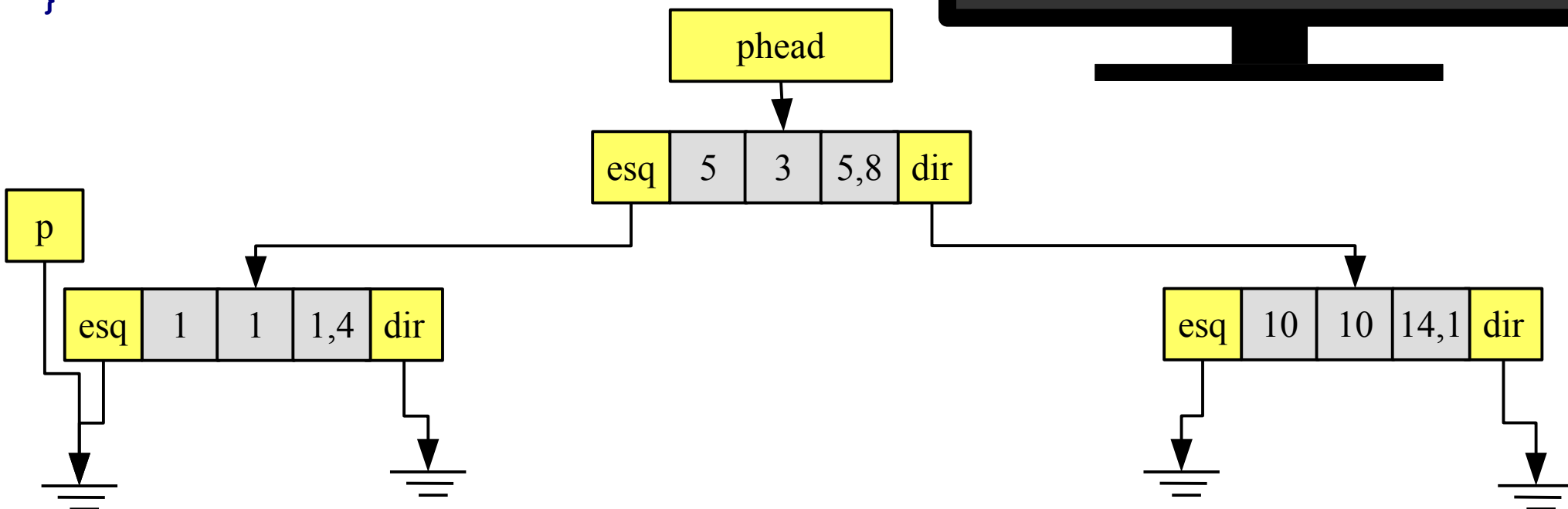
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        → imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



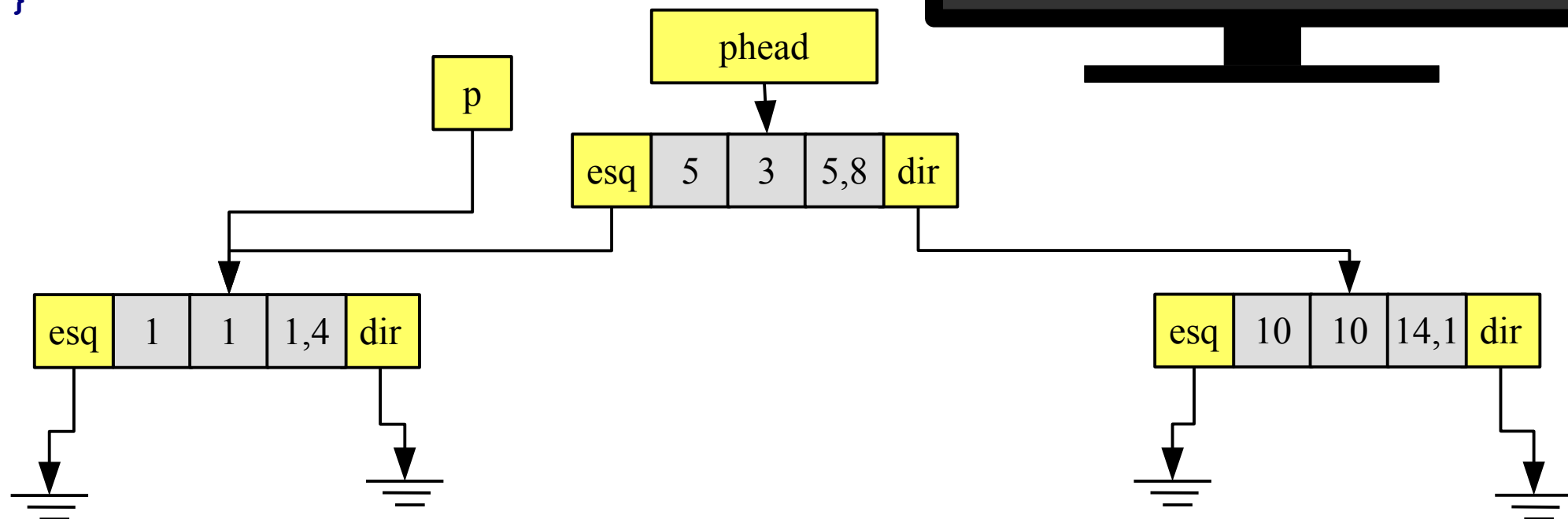
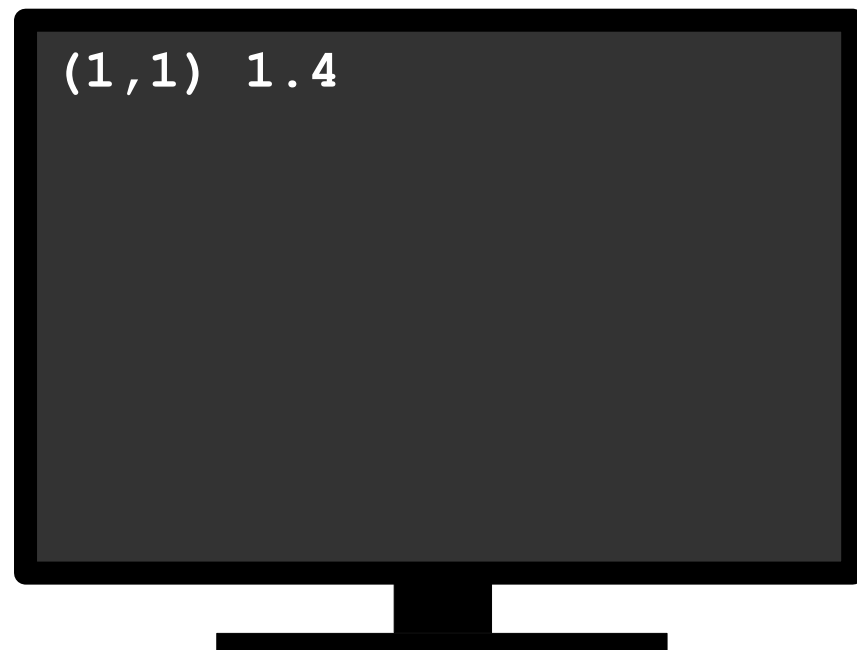
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



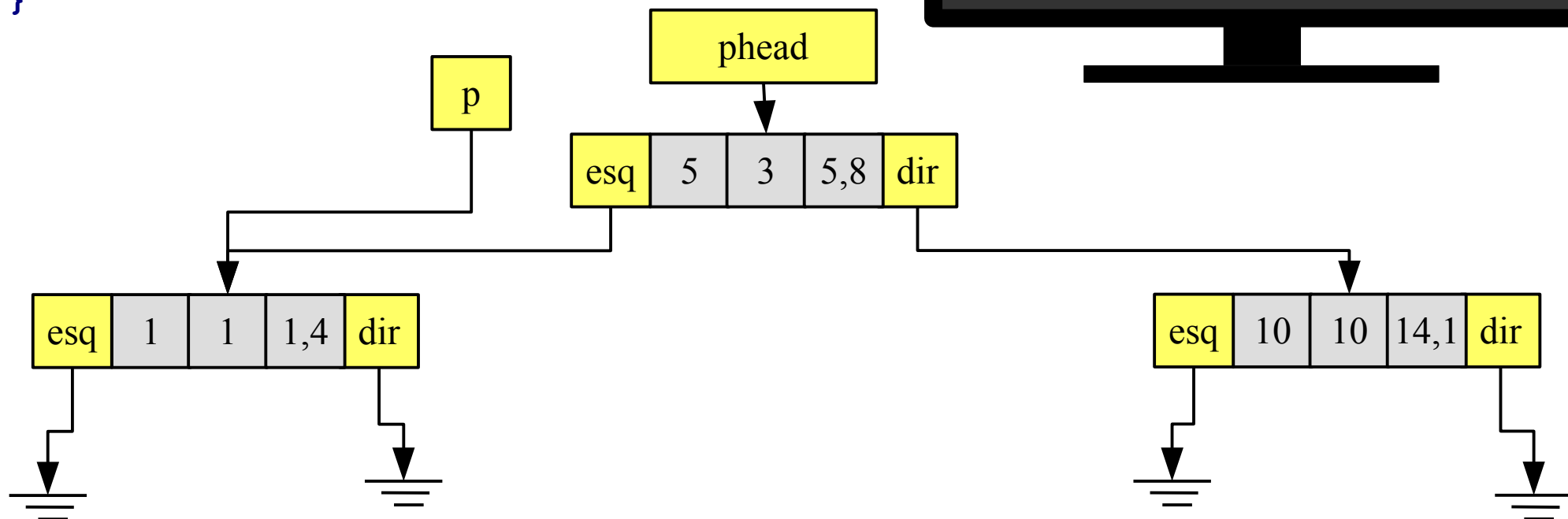
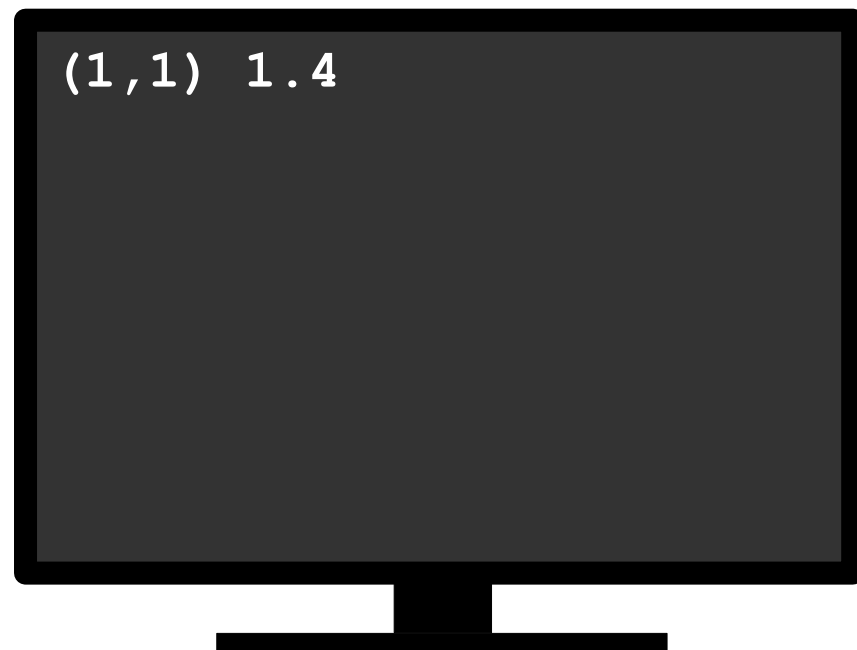
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        → imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



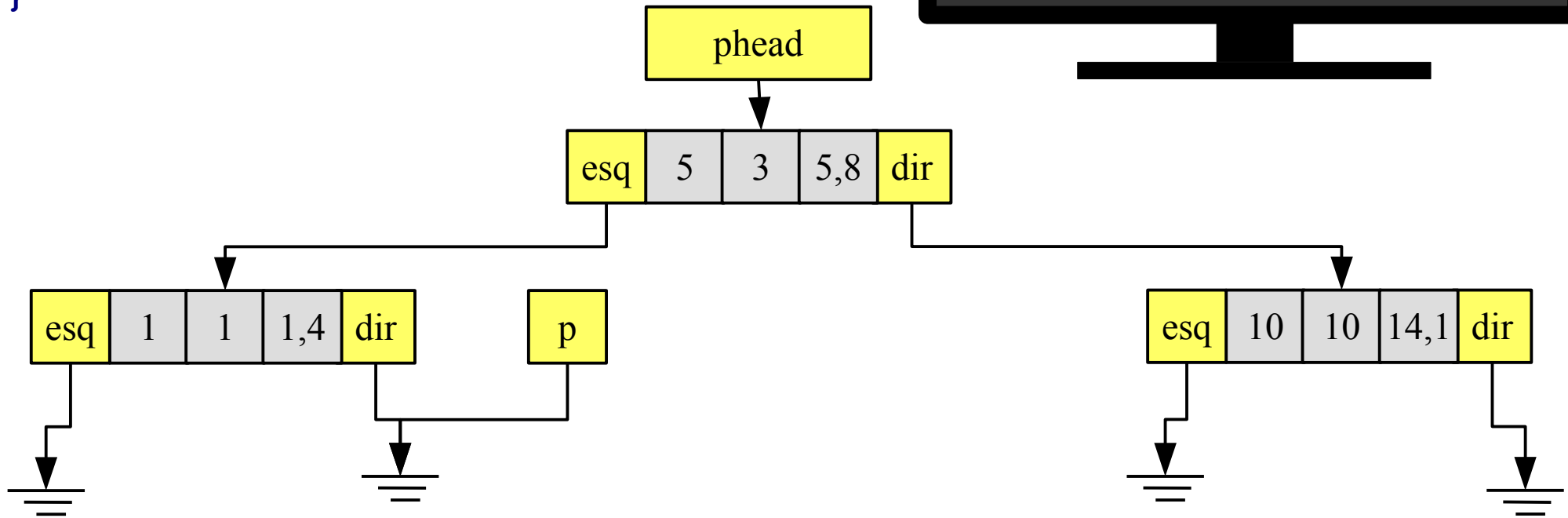
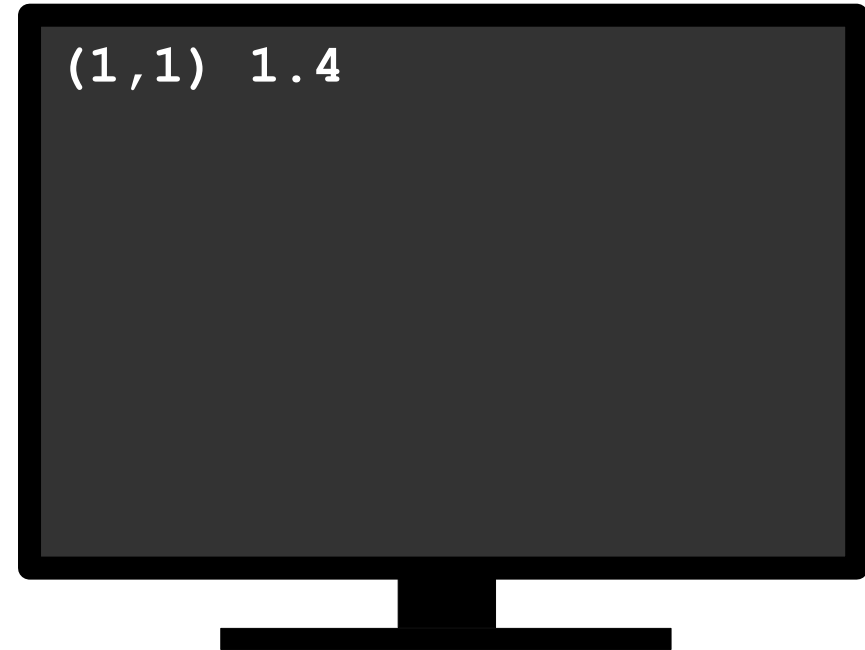
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



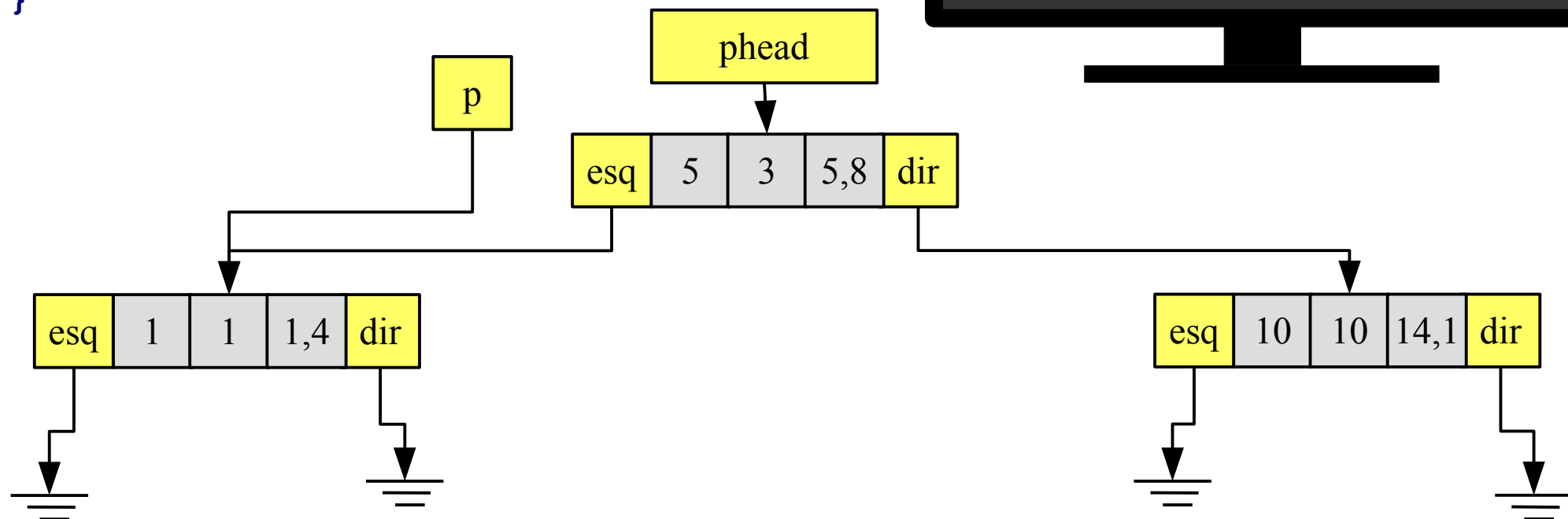
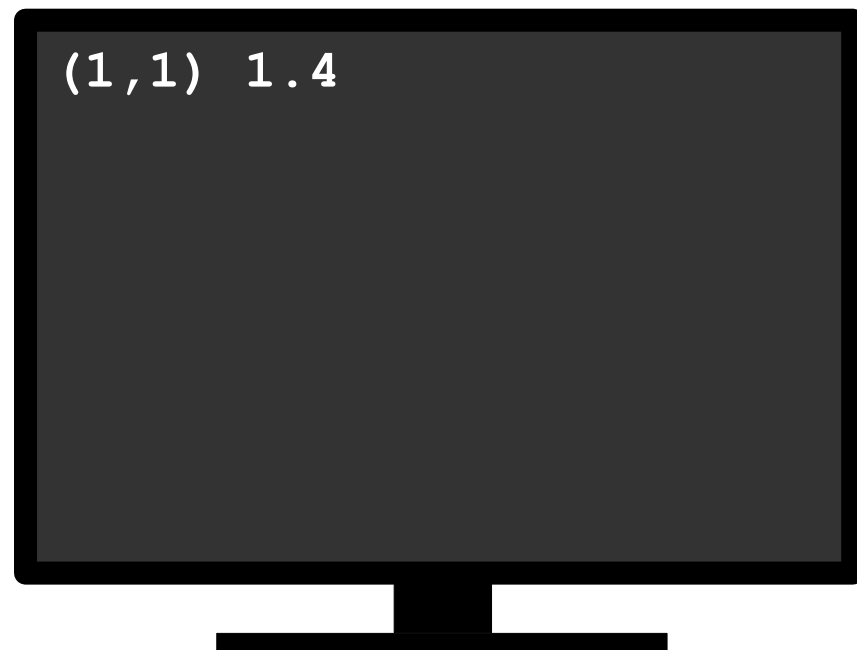
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



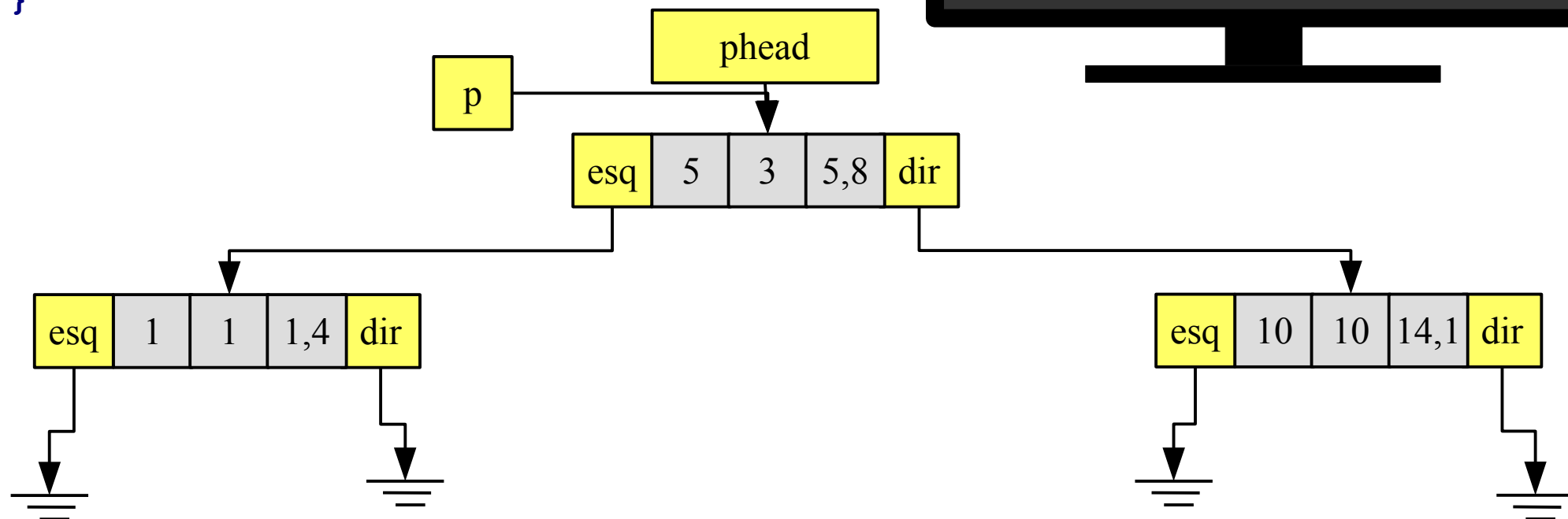
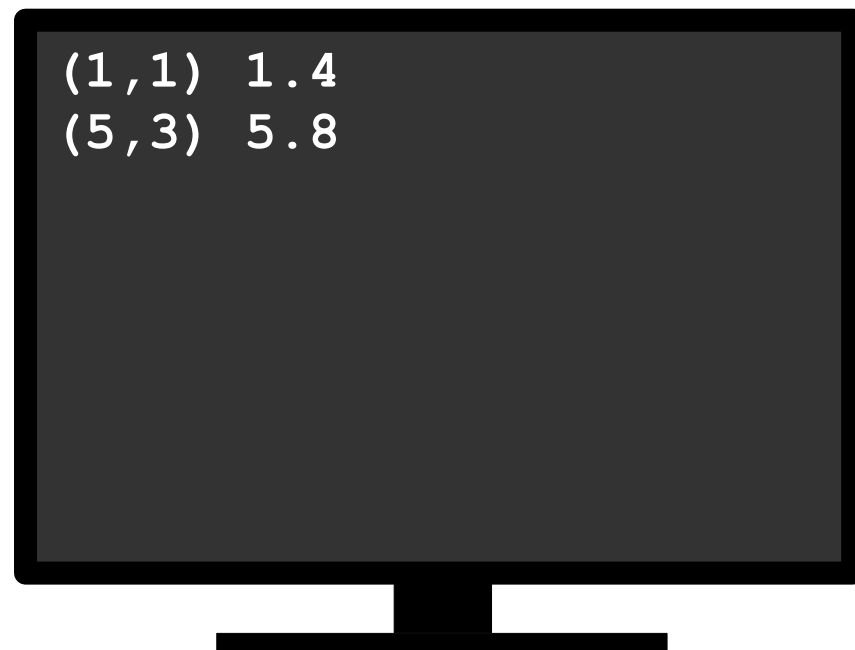
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



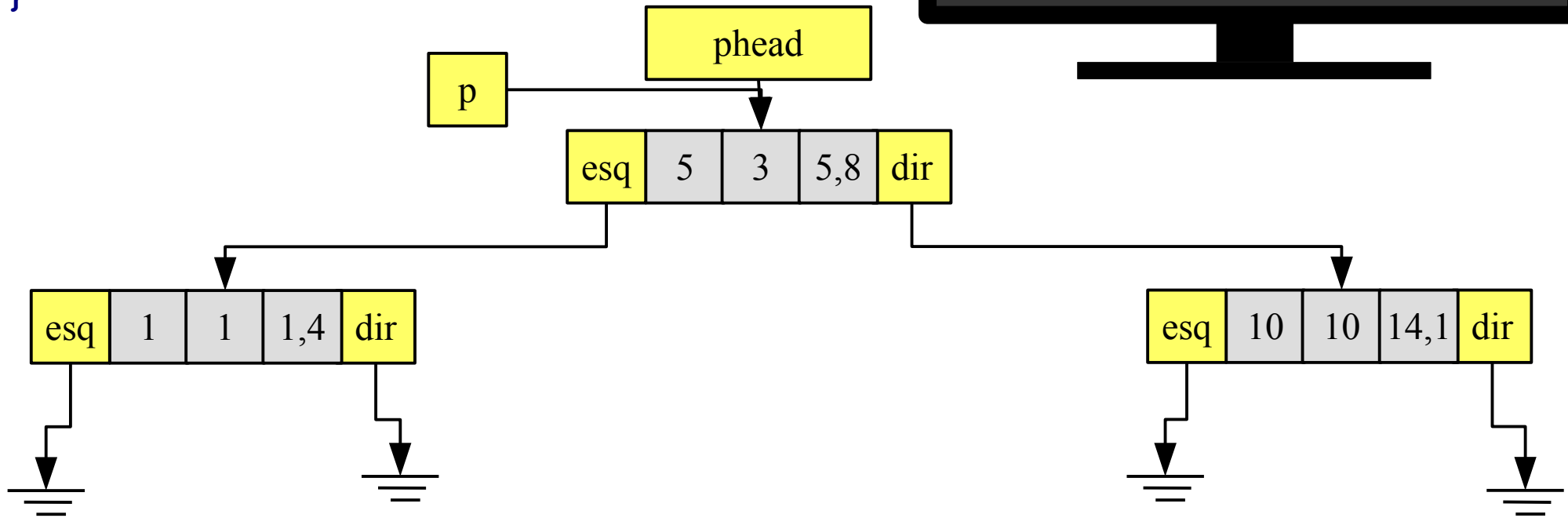
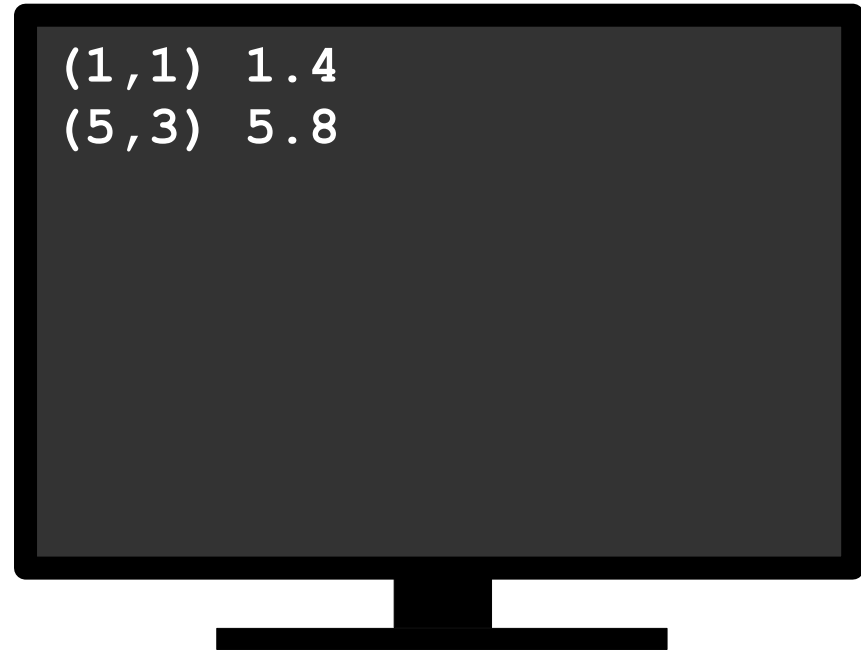
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        → imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



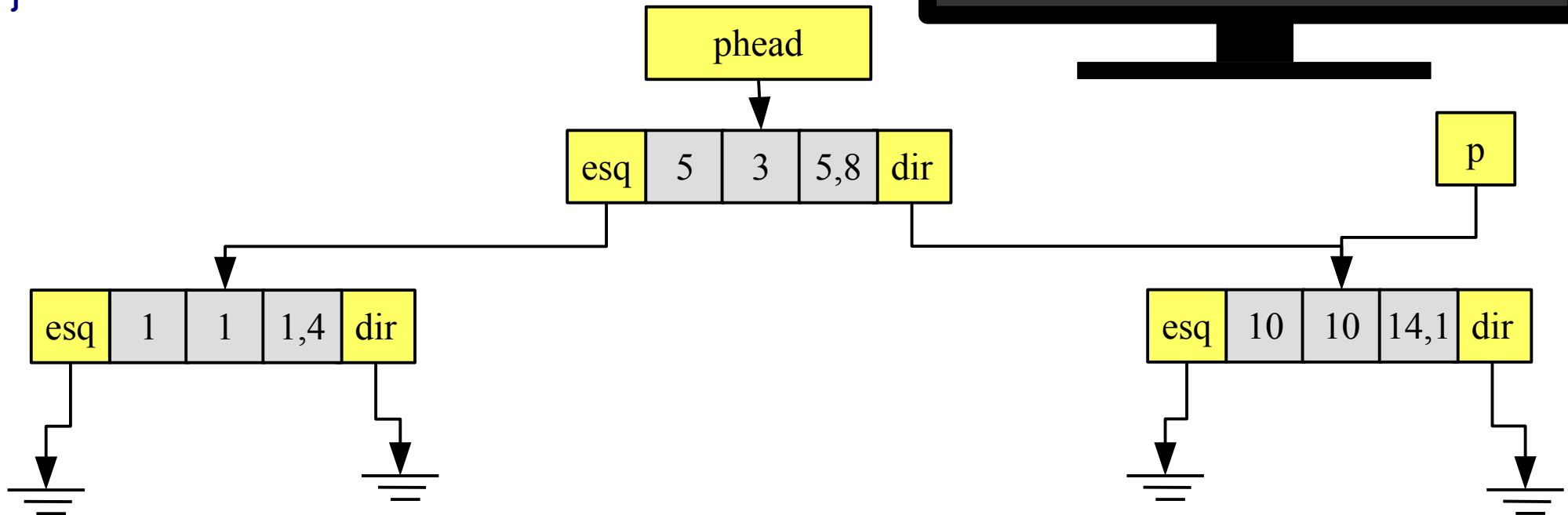
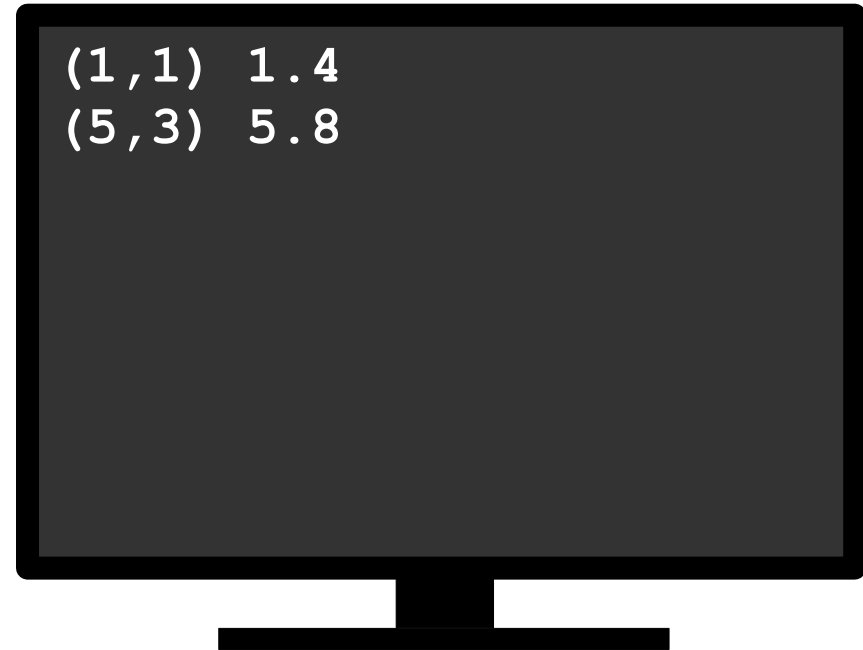
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        → imprimeArvoreBin(p->dir) ;  
    }  
}
```



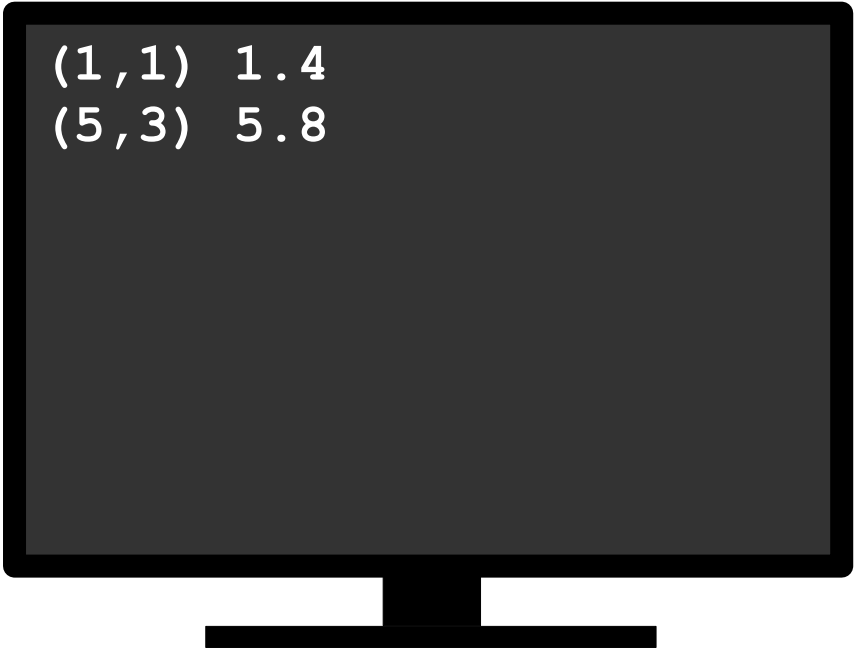
Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        → imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```

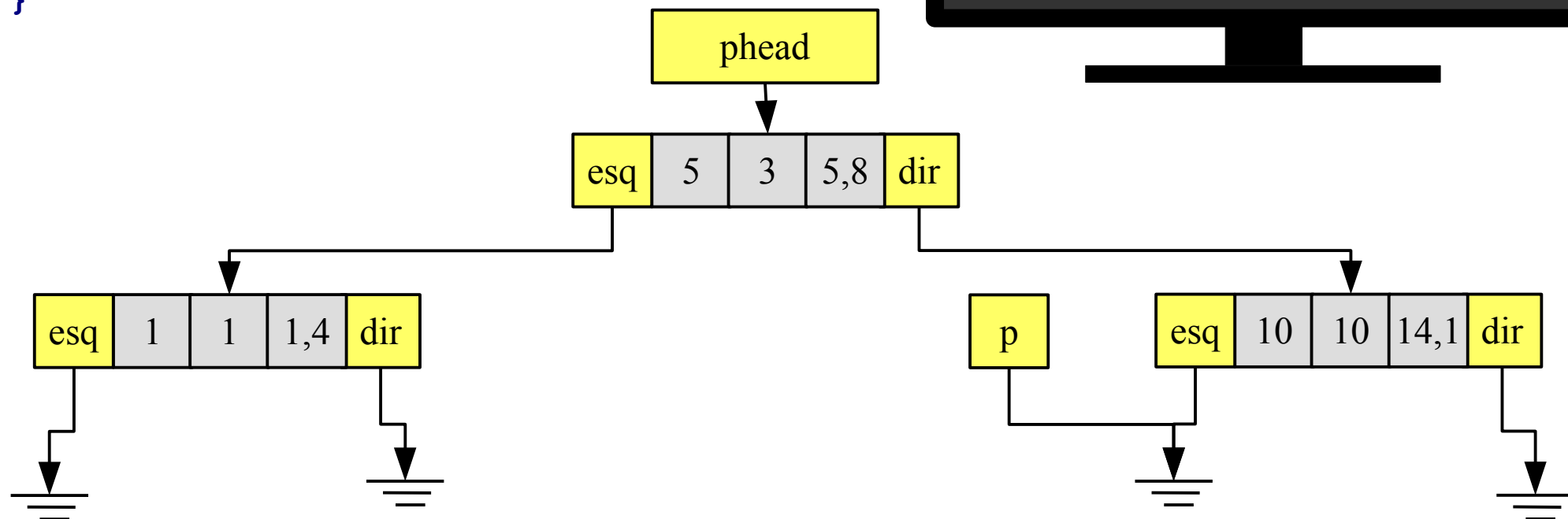


Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```

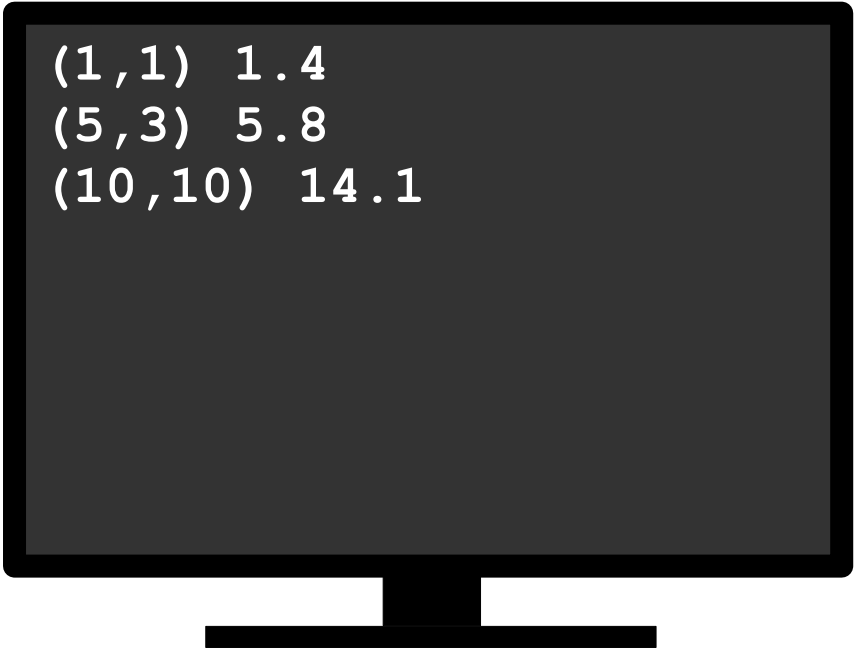


(1,1) 1.4
(5,3) 5.8

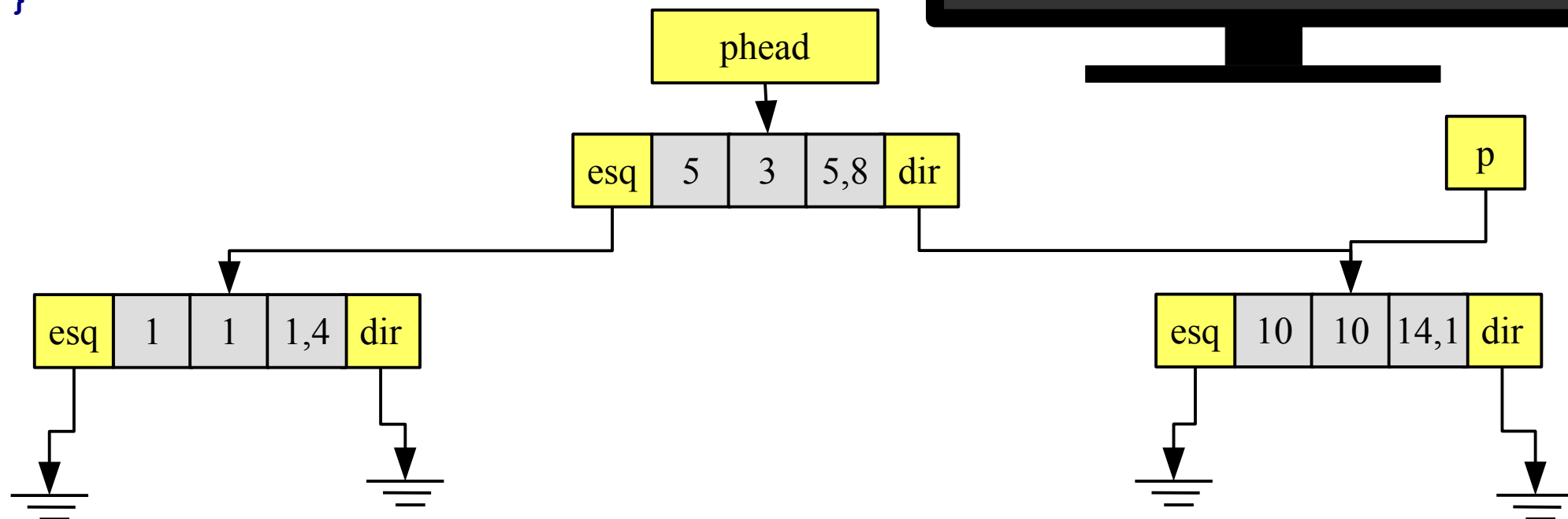


Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        → imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```

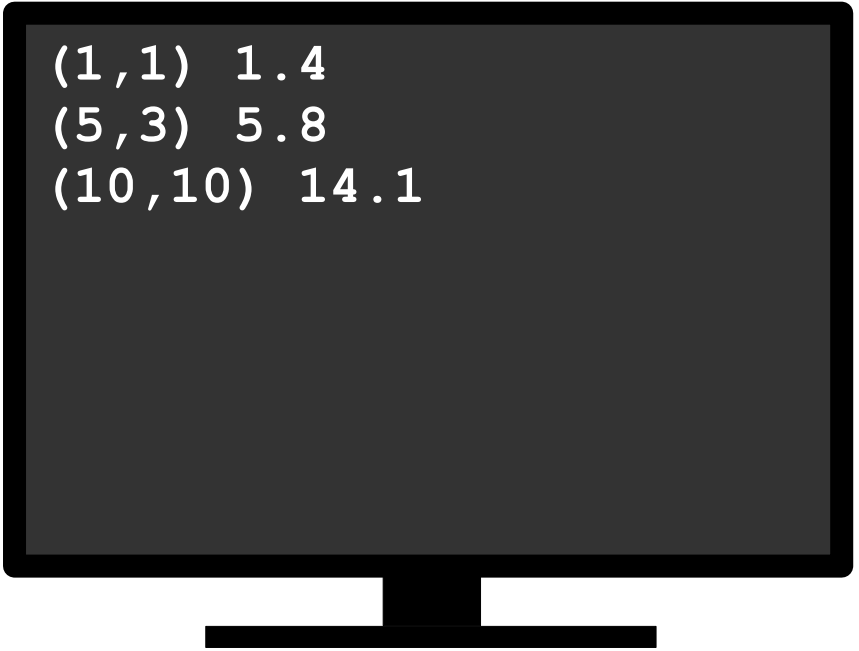


(1,1) 1.4
(5,3) 5.8
(10,10) 14.1

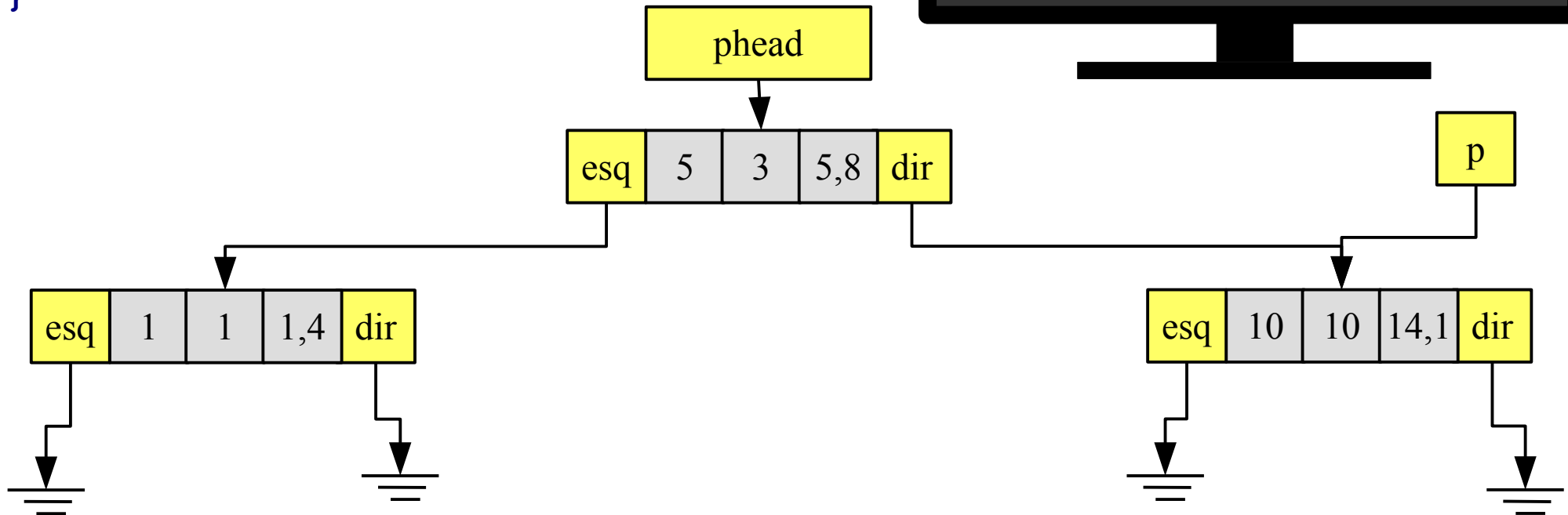


Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        → imprimeArvoreBin(p->dir) ;  
    }  
}
```

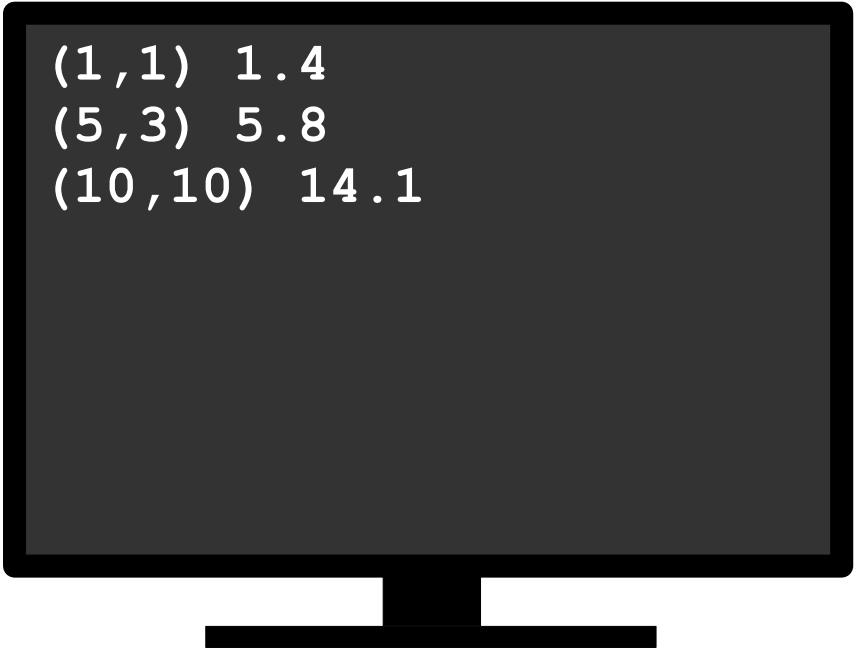


(1,1) 1.4
(5,3) 5.8
(10,10) 14.1

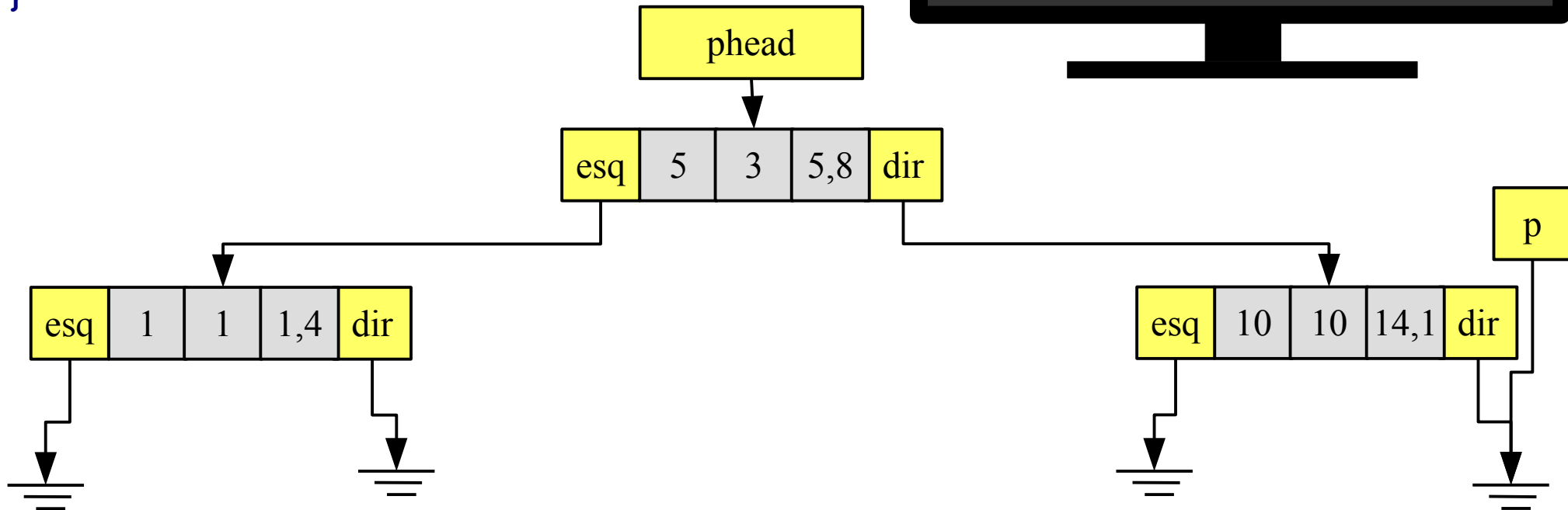


Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        → imprimeArvoreBin(p->dir) ;  
    }  
}
```

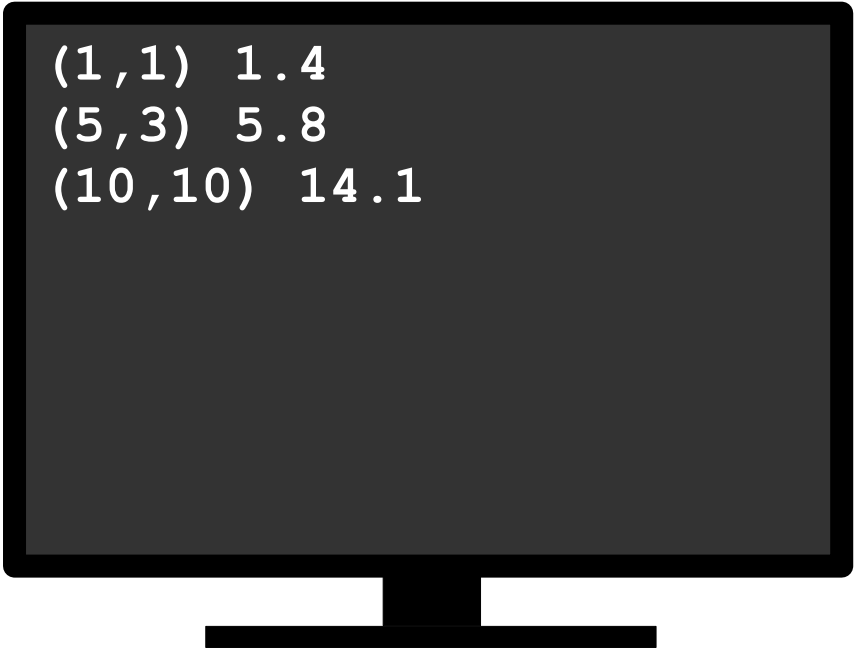


```
(1,1) 1.4  
(5,3) 5.8  
(10,10) 14.1
```

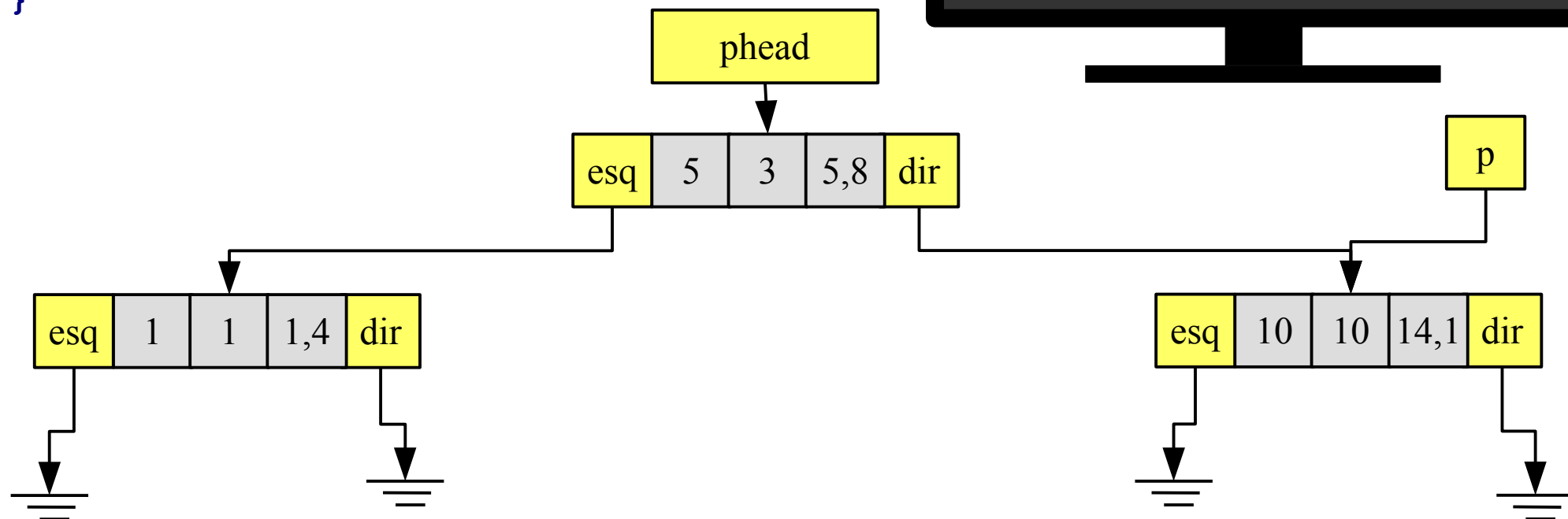


Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```

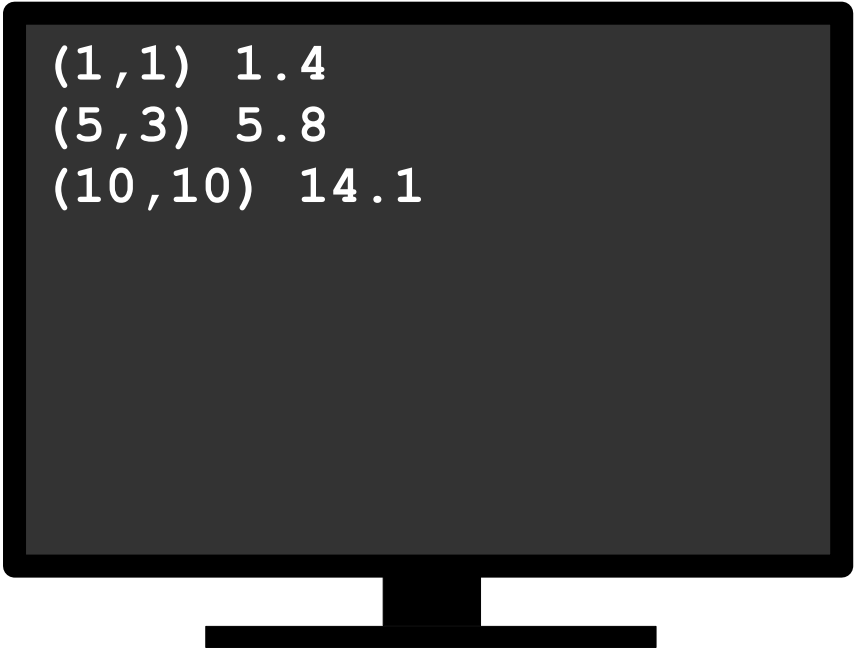


(1,1) 1.4
(5,3) 5.8
(10,10) 14.1

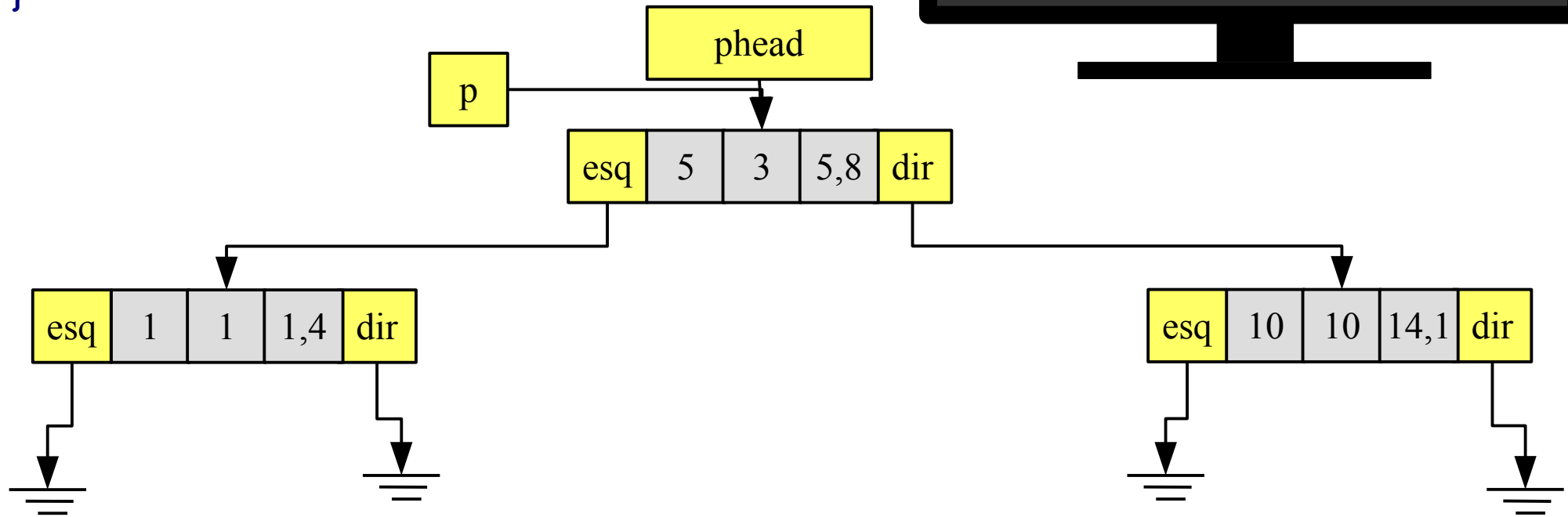


Imprimindo a árvore

```
imprimeArvoreBin(phead) ;  
void imprimeArvoreBin(struct coord * p)  
{  
    if (p)  
    {  
        imprimeArvoreBin(p->esq) ;  
        imprimeElemento(p) ;  
        imprimeArvoreBin(p->dir) ;  
    }  
}
```



```
(1,1) 1.4  
(5,3) 5.8  
(10,10) 14.1
```



Buscando Elemento

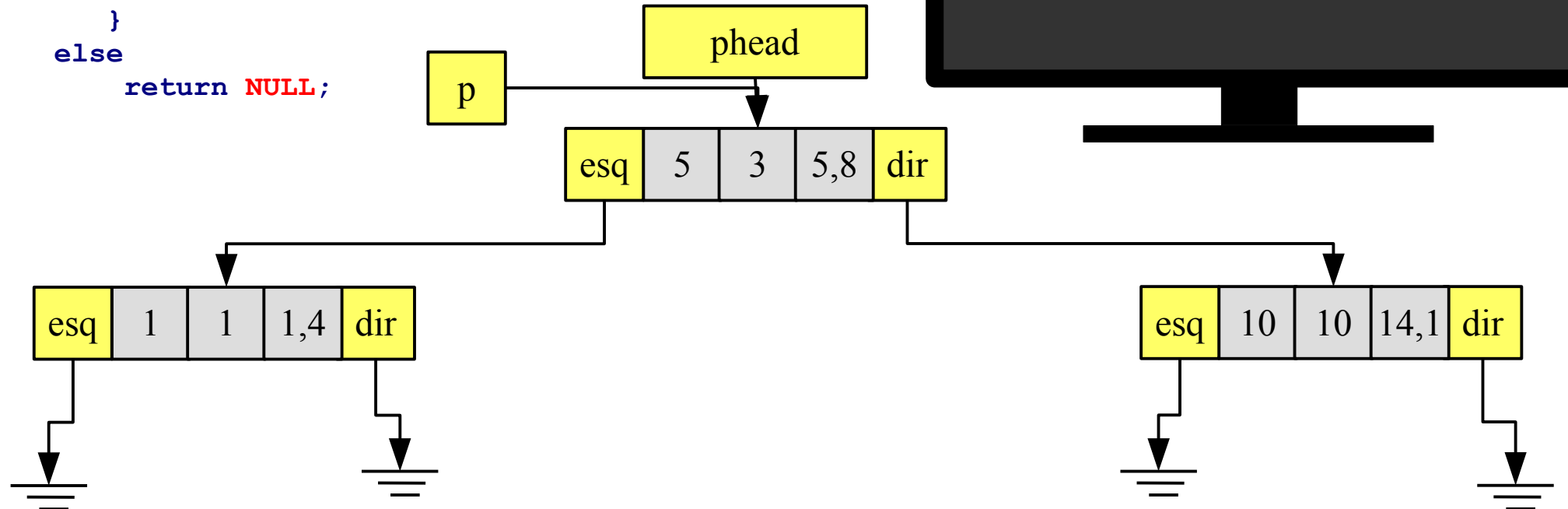
```
struct coord * buscaCoord(int x, int y, struct coord * p)
{
    if (p)
        if ((x==p->x) && (y==p->y))
            return p;
        else
        {
            float d = distEuclid(x,y);
            if (d > p->d)
                return buscaCoord(x, y, p->dir);
            else
                return buscaCoord(x, y, p->esq);
        }
    else
        return NULL;
}
```

Buscando Elemento

```
imprimeElemento(buscaCoord(1,1,phead)) ;
```

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

```
{  
    if (p)  
        if ((x==p->x) && (y==p->y))  
            return p;  
        else  
        {  
            float d = distEuclid(x,y);  
            if (d > p->d)  
                return buscaCoord(x, y, p->dir);  
            else  
                return buscaCoord(x, y, p->esq);  
        }  
    else  
        return NULL;  
}
```



Buscando Elemento

```
imprimeElemento (buscaCoord(1,1,phhead)) ;
```

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

{

```
if (p)
```

```
if ( (x==p->x) && (y==p->y) )
```

```
return p;
```

else

{

```
float d = distEuclid(x,y);
```

```
if (d > p->d)
```

```
return buscaCoord(x, y, p->dir);
```

else

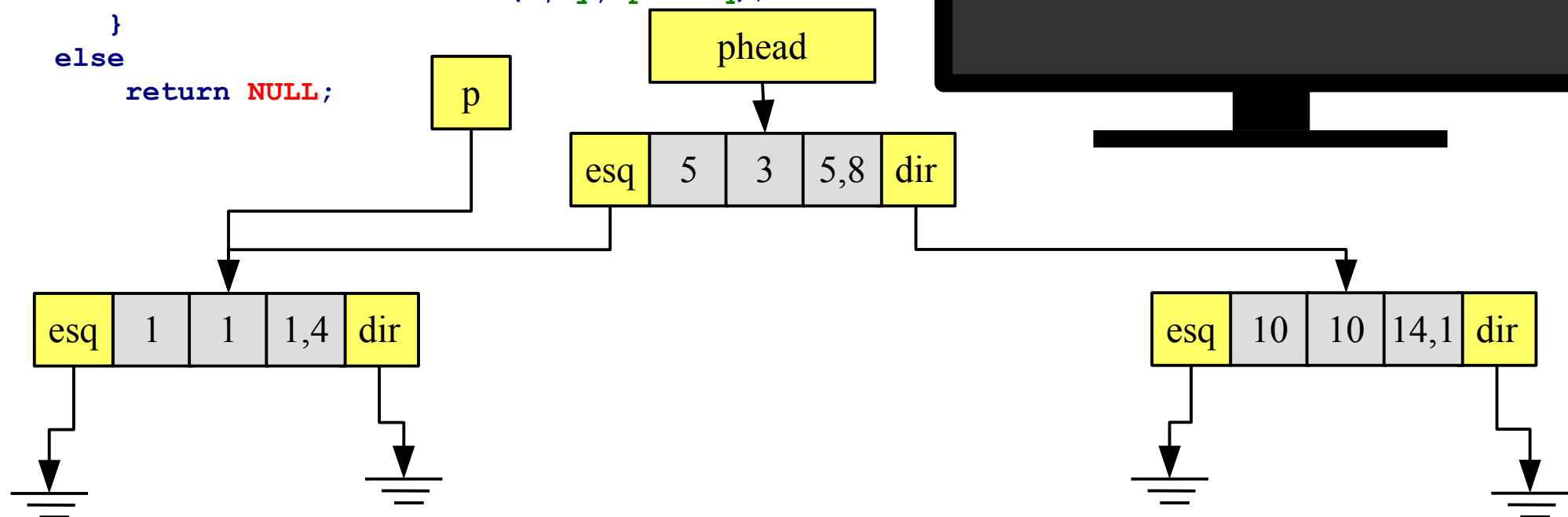
```
return buscaCoord(x, y, p->esq);
```

}

else

```
return NULL;
```

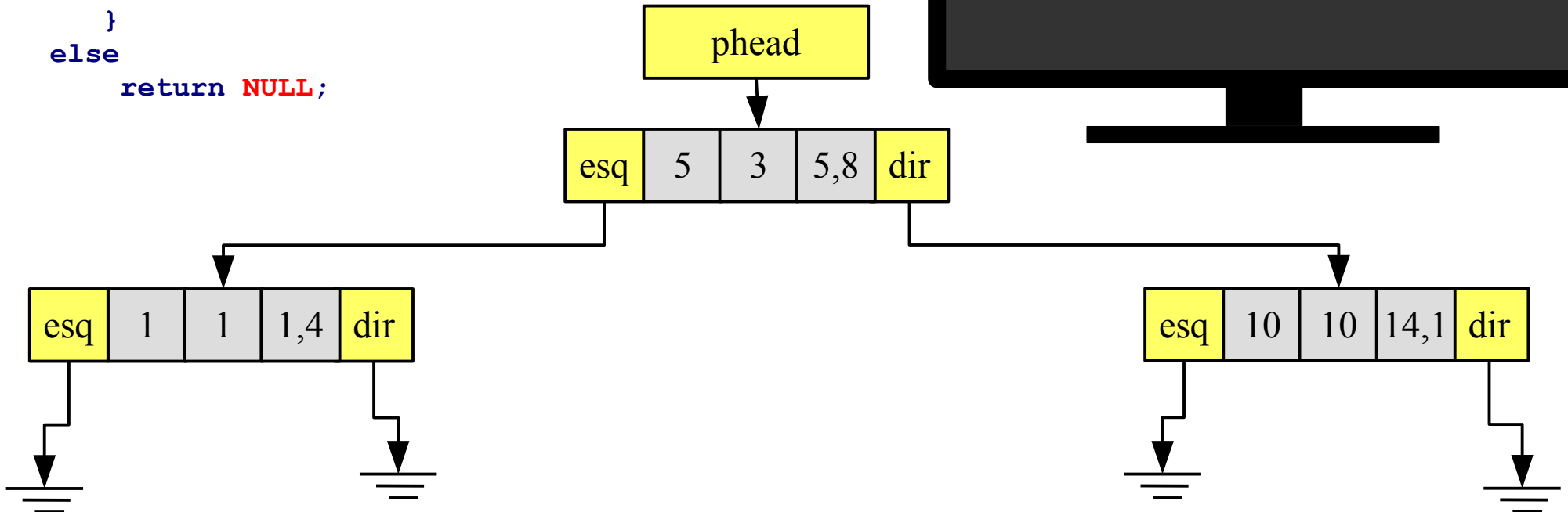
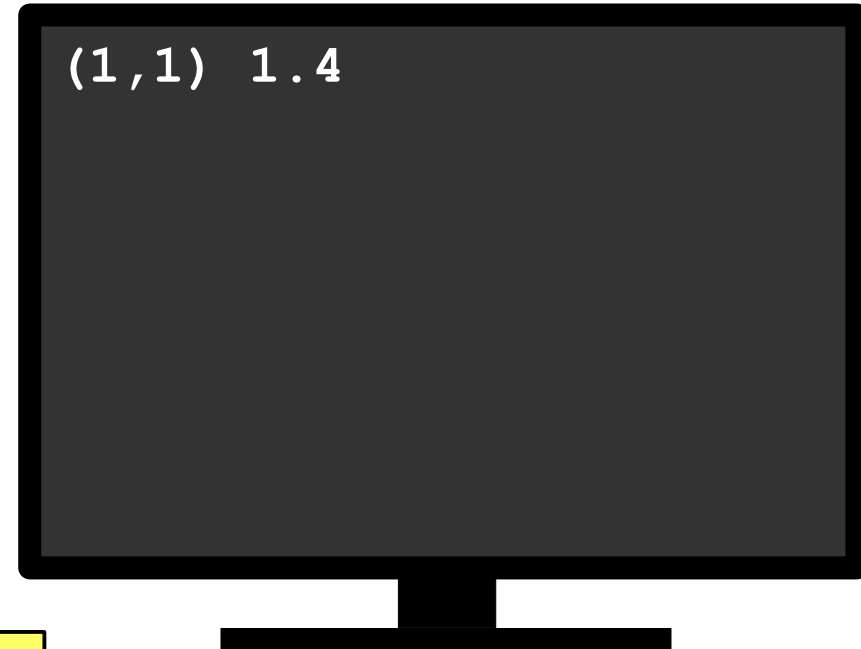
}



Buscando Elemento

```
imprimeElemento(buscaCoord(1,1,phead)) ; ←
```

```
struct coord * buscaCoord(int x, int y, struct coord * p)
{
    if (p)
        if ((x==p->x) && (y==p->y))
            return p;
        else
        {
            float d = distEuclid(x,y);
            if (d > p->d)
                return buscaCoord(x, y, p->dir);
            else
                return buscaCoord(x, y, p->esq);
        }
    else
        return NULL;
}
```



Exclusão em Árvores Binárias

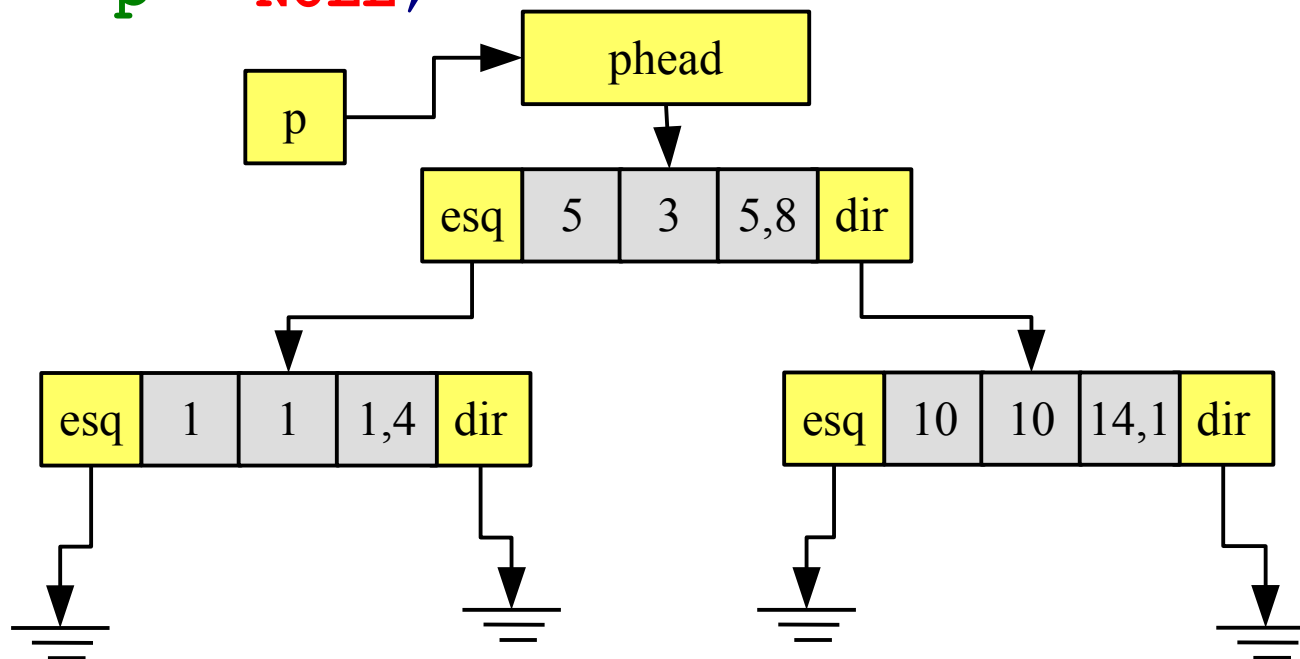
Excluindo Toda a Árvore

```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```


Excluindo Toda a Árvore

`limparArvoreBin(&phead)`

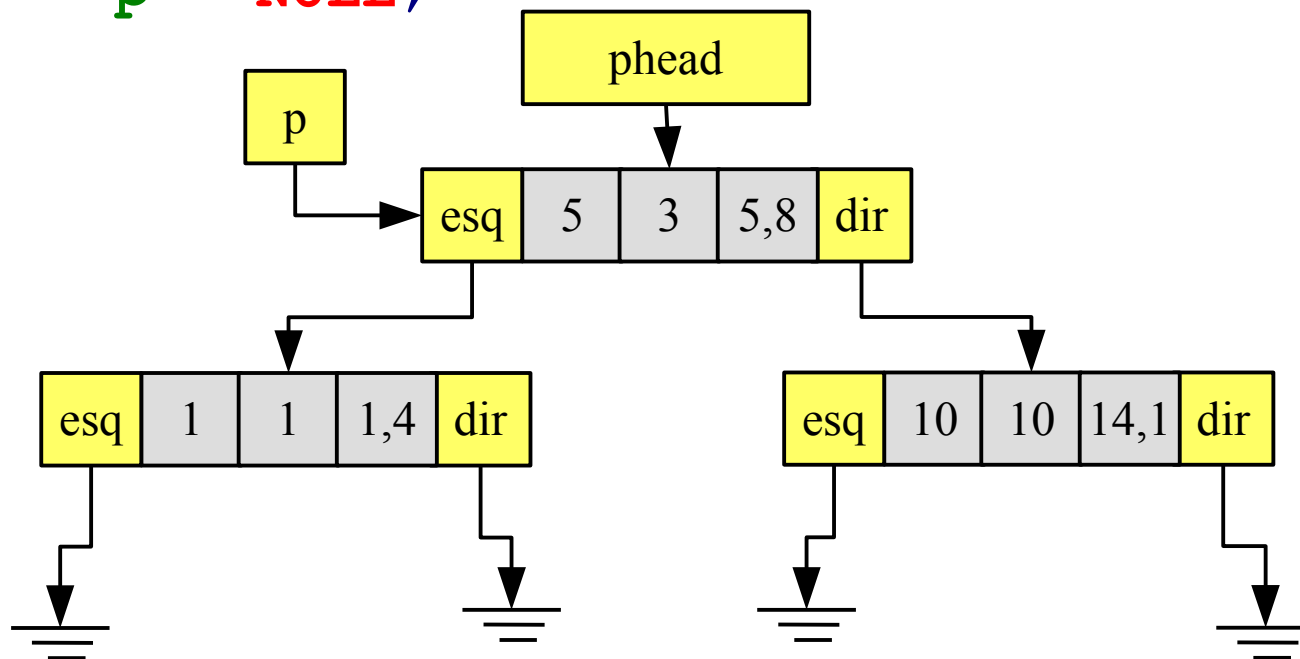
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        → limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

`limparArvoreBin(&phead)`

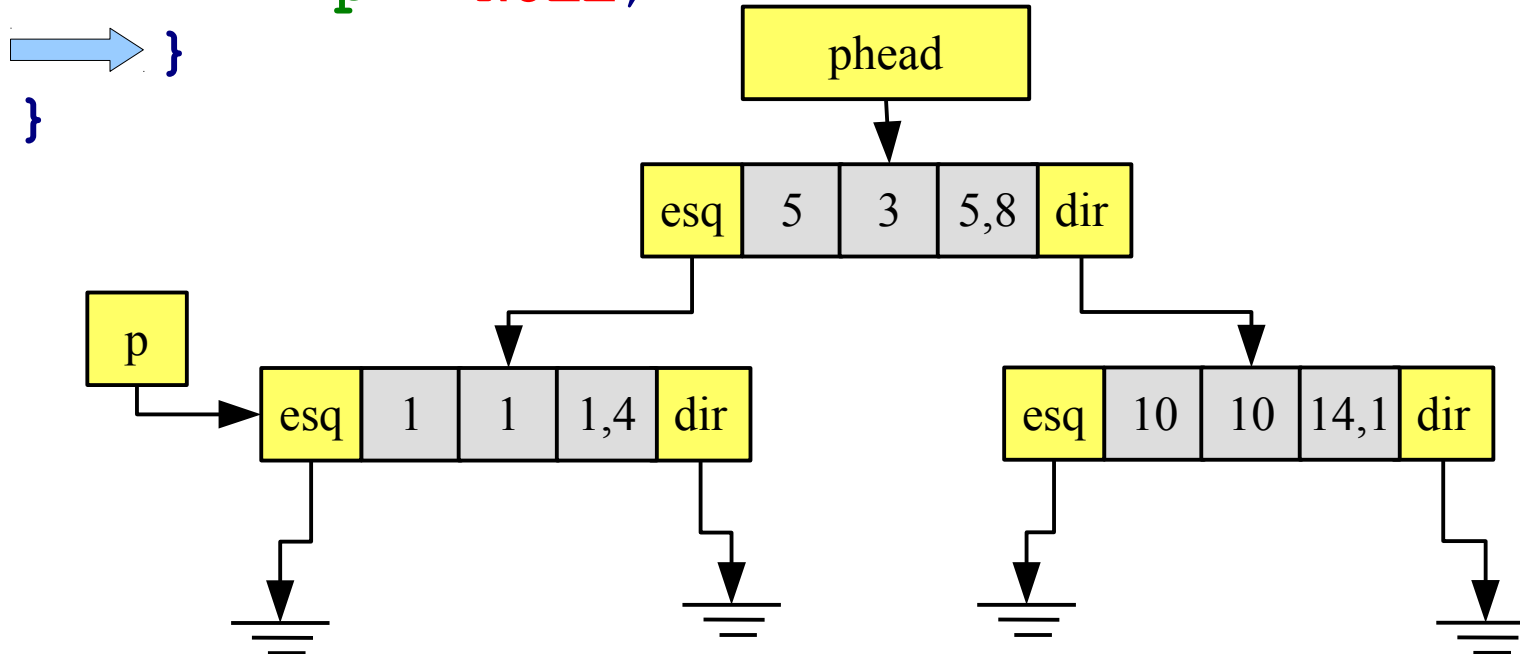
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        → limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

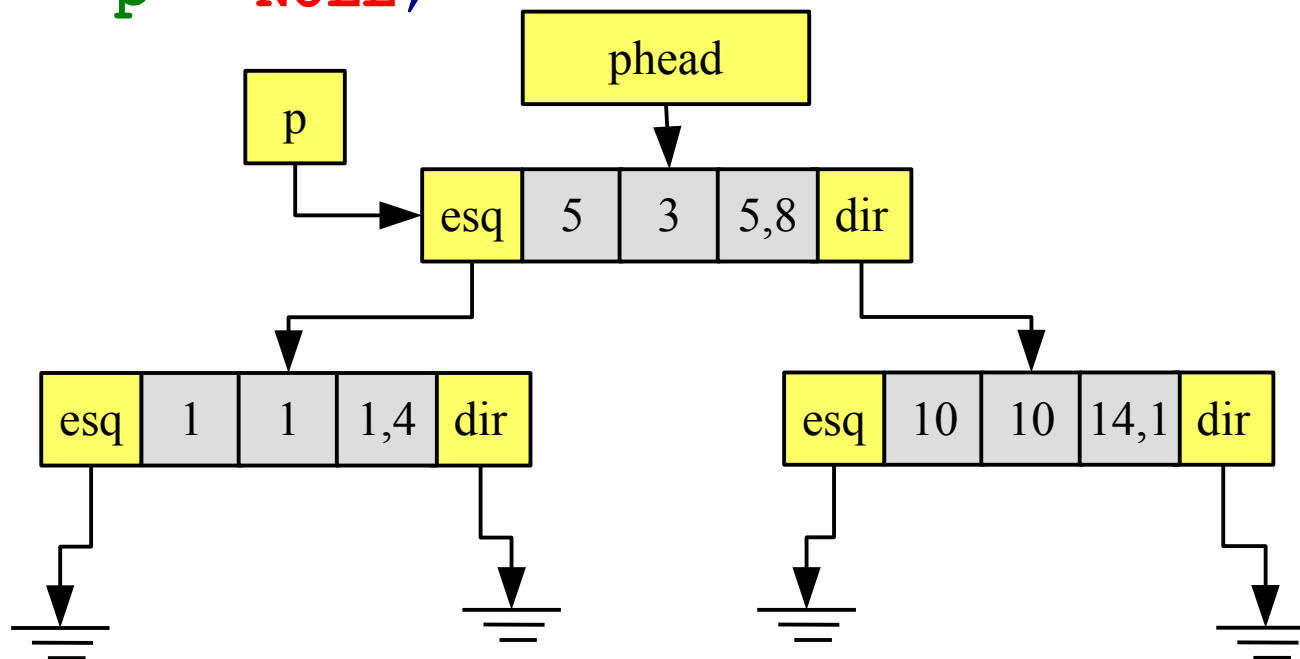
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

`limparArvoreBin(&phead)`

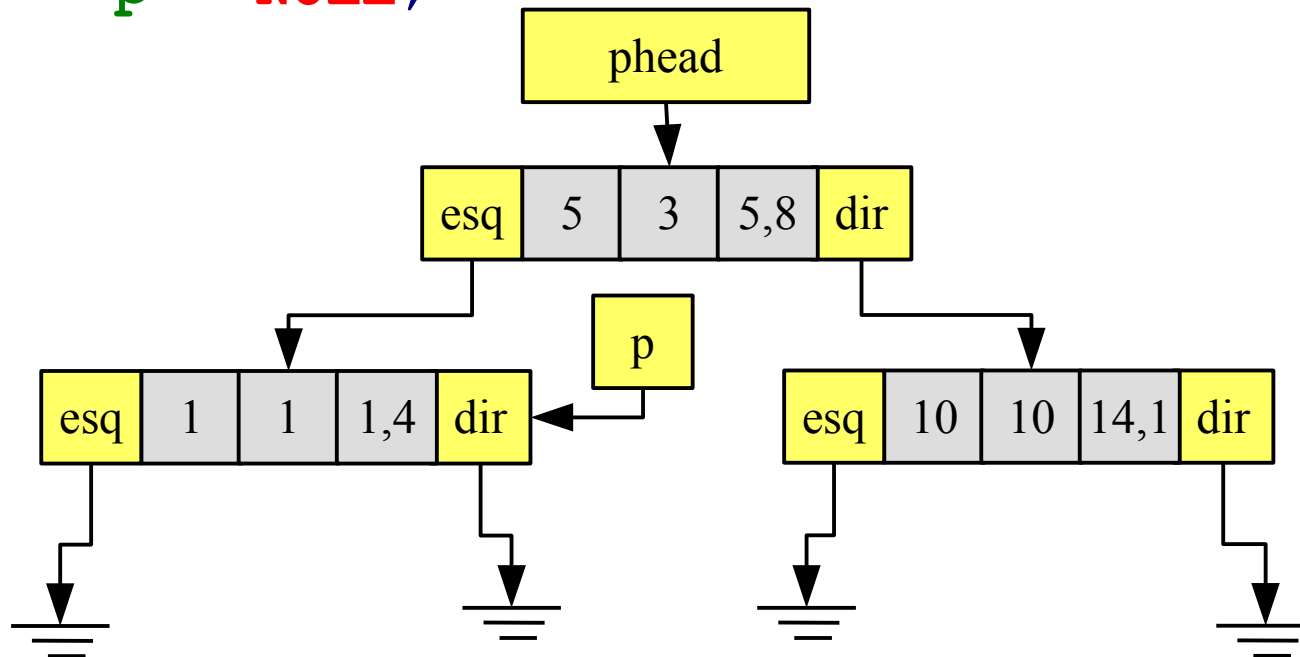
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

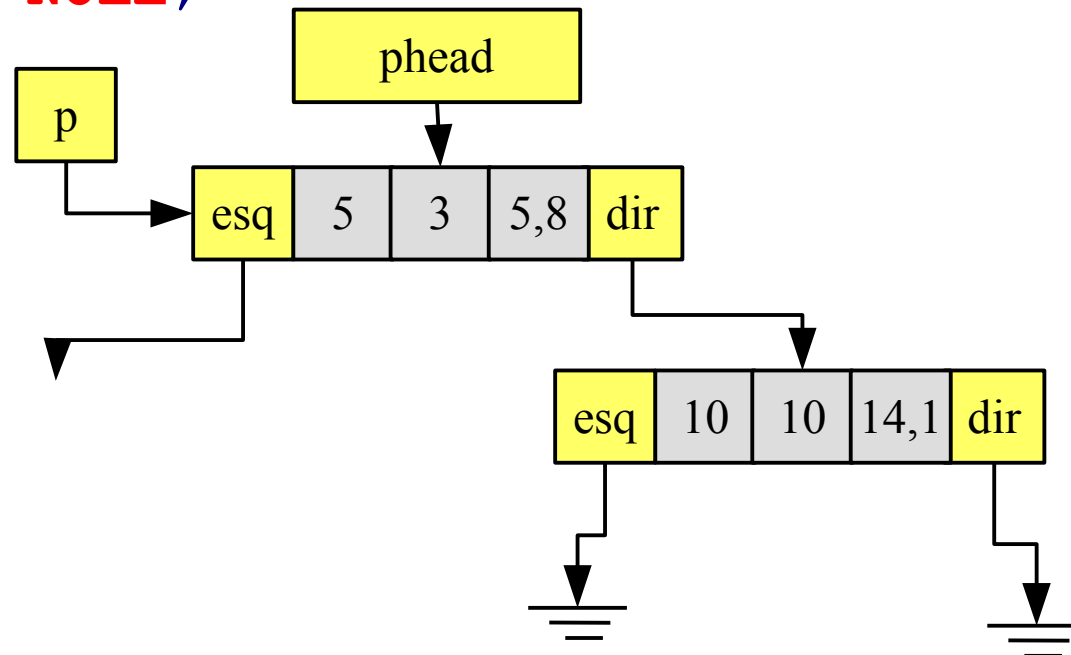
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

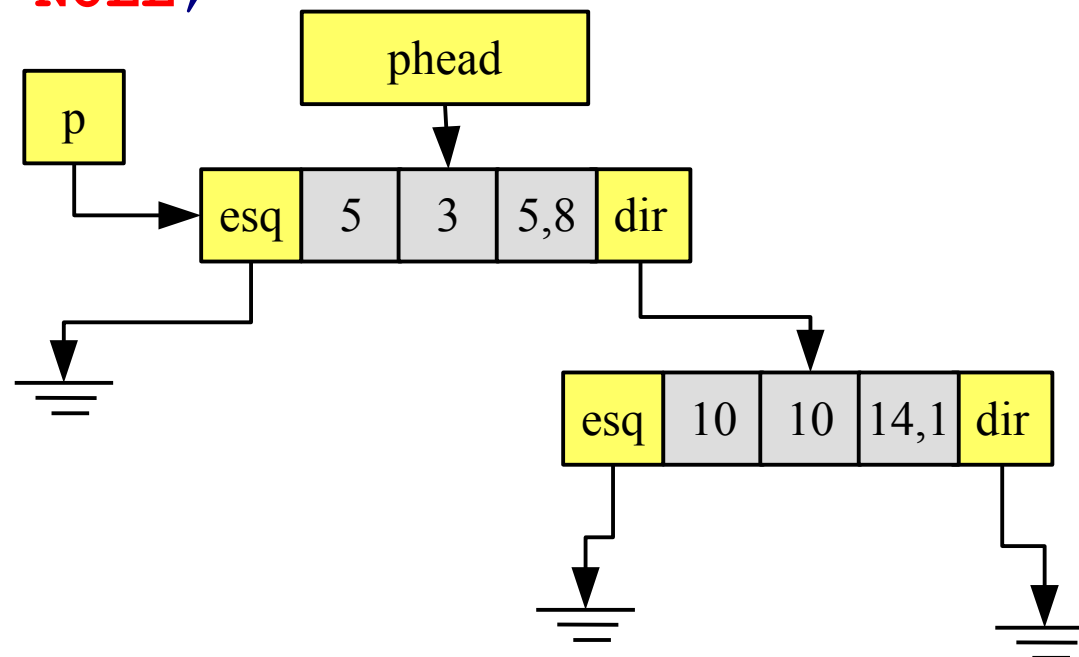
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

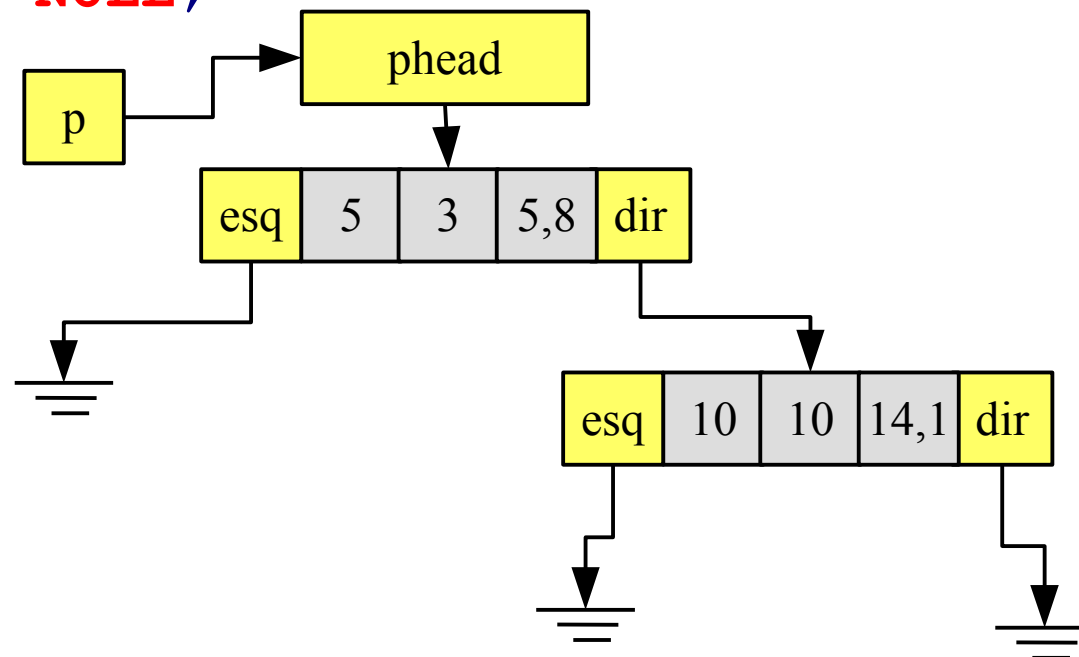
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

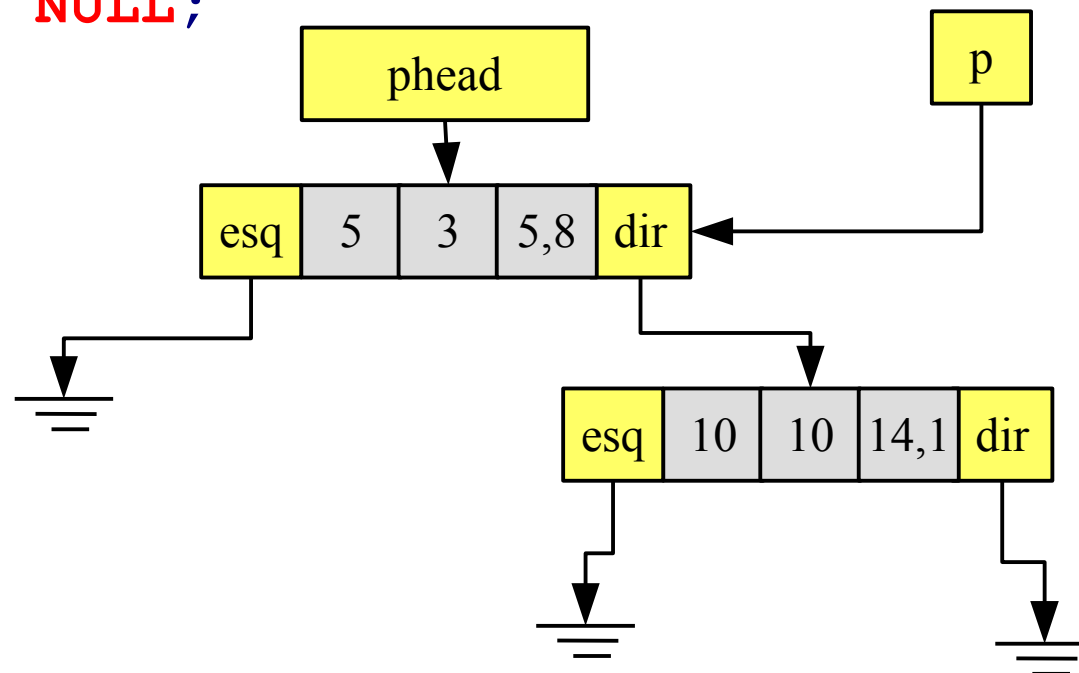
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

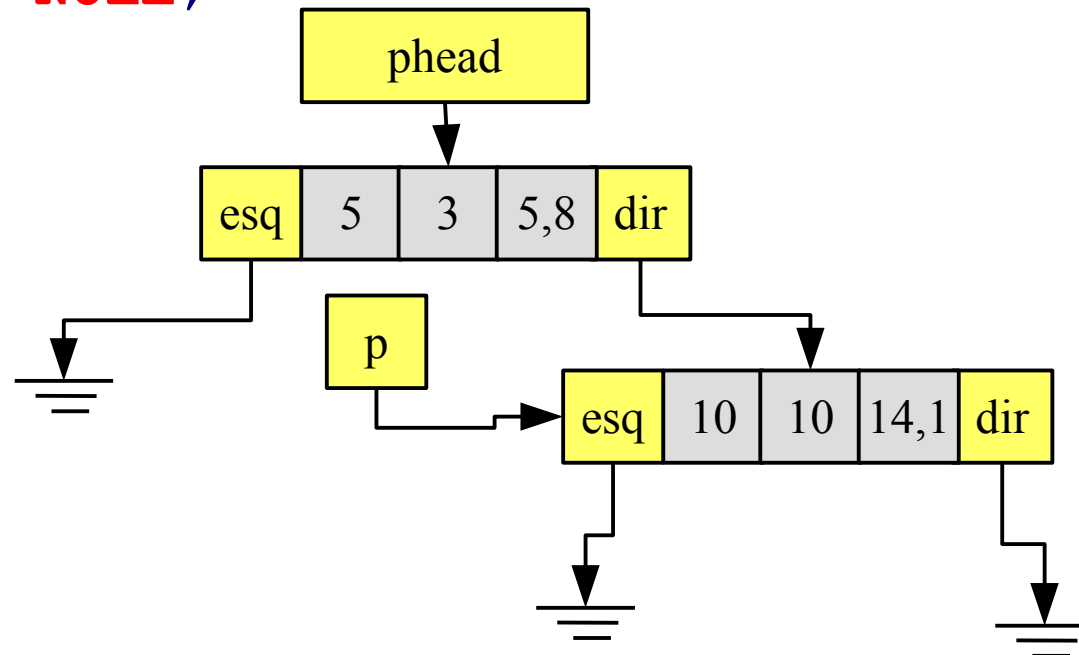
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

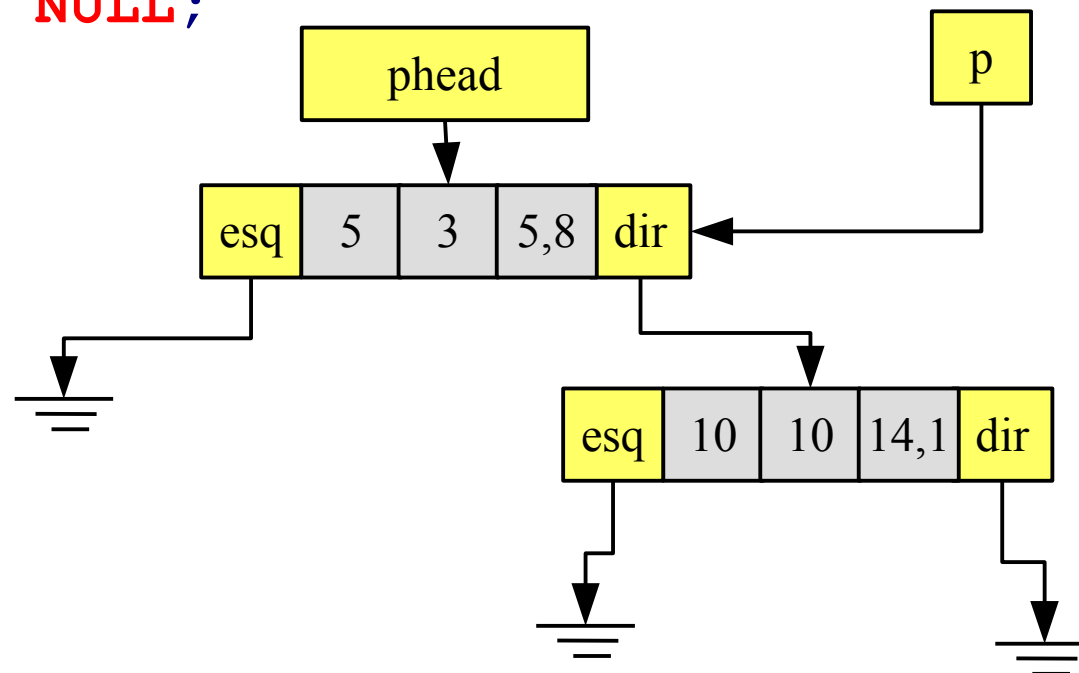
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

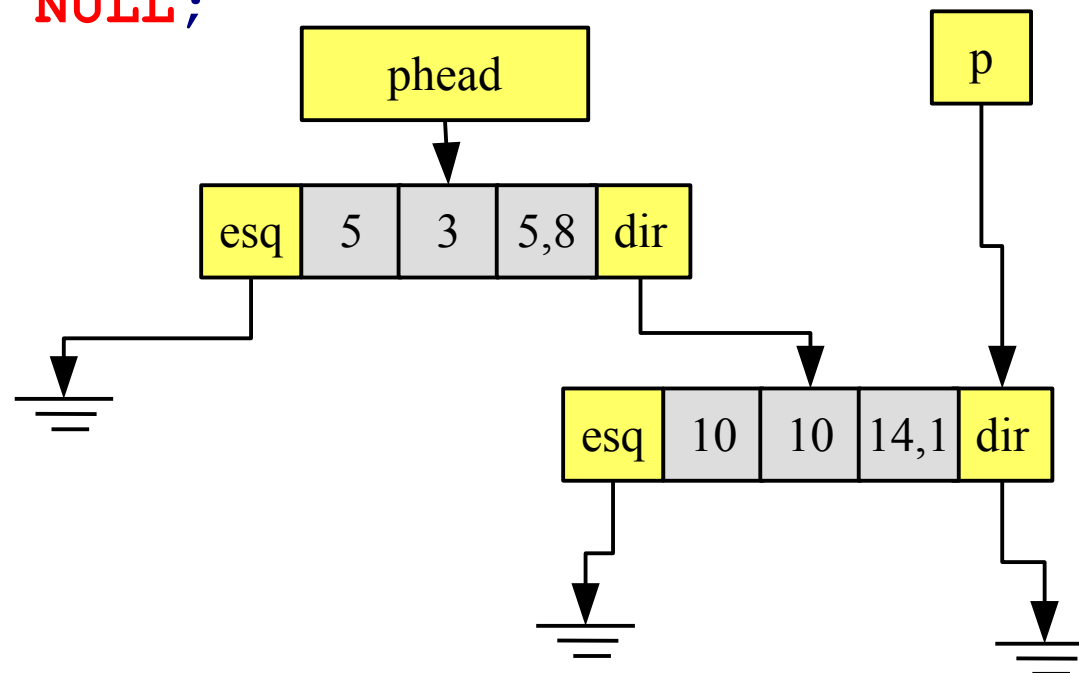
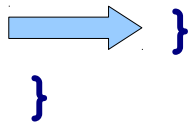
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

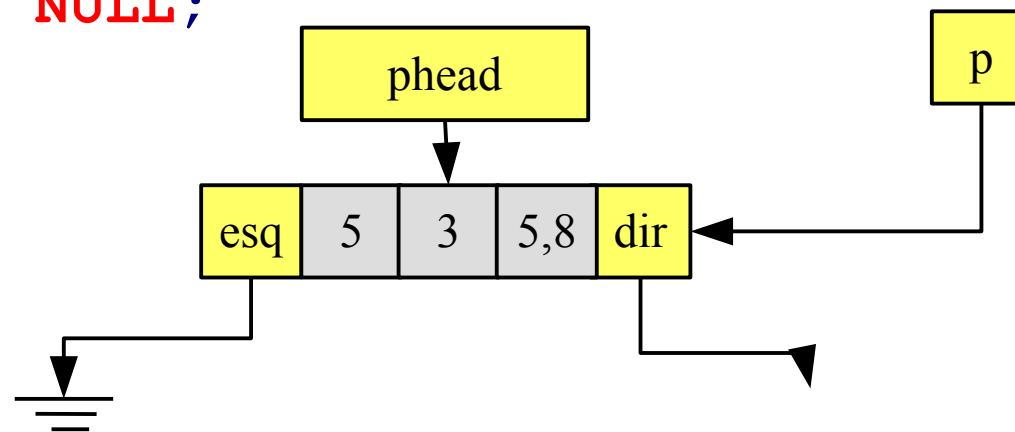
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

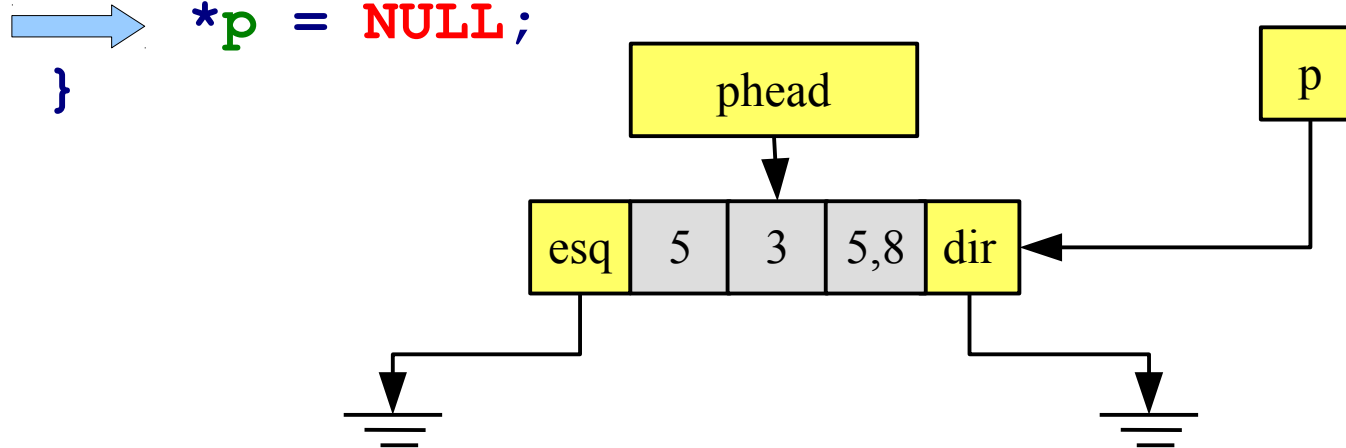
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

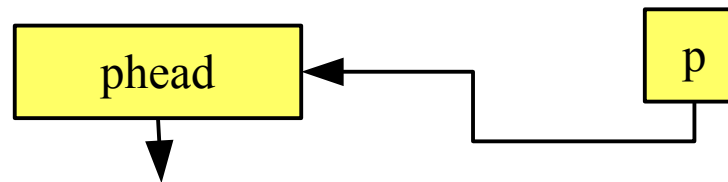
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin(&phead)
```

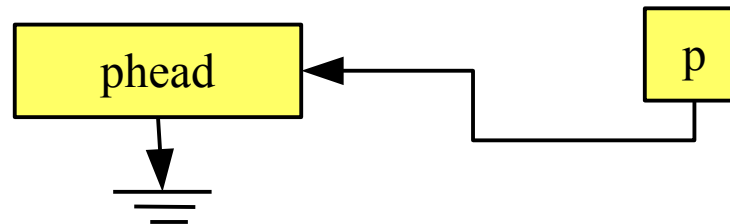
```
void limparArvoreBin(struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin(&((*p)->esq));
        limparArvoreBin(&((*p)->dir));
        → libera(*p);
        *p = NULL;
    }
}
```



Excluindo Toda a Árvore

```
limparArvoreBin (&phead)
```

```
void limparArvoreBin (struct coord ** p)
{
    if (*p)
    {
        limparArvoreBin (&((*p)->esq));
        limparArvoreBin (&((*p)->dir));
        libera (*p);
        *p = NULL;
    }
}
```



Usando isso tudo!

```
int main()
{
    struct coord * phead = NULL;
    int i;
    //Insere 5 elementos na árvore
    for (i=0; i<5; i++)
        insereArvoreBin(&phead) ;

    imprimeArvoreBin(phead) ;

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

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

    limparArvoreBin(&phead) ;
    imprimeArvorebin(phead) ;
}
```

Fim