



# Lista Encadeada

# Definição de Lista Encadeada

- Sequência de estruturas (**nós** da lista) ligados entre si por ponteiros.
- Esta sequência pode ser acessada por um ponteiro para o primeiro nó (cabeça da lista).
- Cada nó contém um ponteiro que aponta para a estrutura que é a sua sucessora na lista.

# Definição de Lista Encadeada

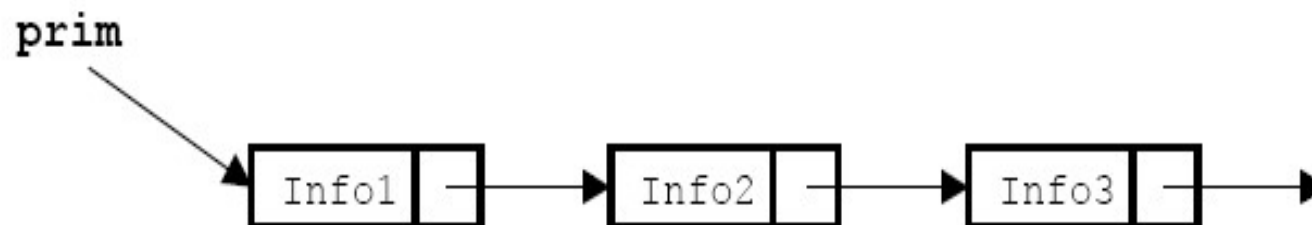
- O ponteiro da última estrutura da lista aponta para NULL, indicando que chegou ao final da lista.
- Esta estrutura de dados é criada dinamicamente na memória (utiliza-se malloc() e free()), de modo que se torna simples introduzir, retirar ou ordenar nós nela.

# Definição de Lista Encadeada

- Numa lista encadeada, para cada novo elemento inserido na estrutura, alocamos um espaço de memória para armazená-lo.
- Desta forma, o espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenado.
- **Não podemos garantir que os elementos armazenados na lista ocuparão um espaço de memória sequencial, portanto não temos acesso direto aos elementos da lista.**

# Definição de Lista Encadeada

- Para que seja possível percorrer todos os elementos da lista, devemos explicitamente guardar o encadeamento dos elementos, o que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista.



# Definição de Lista Encadeada

- A estrutura consiste numa sequência encadeada de elementos, em geral chamados de *nós da lista*.
- A lista é representada por um ponteiro para o primeiro elemento (ou nó).
- Do primeiro elemento, podemos alcançar o segundo seguindo o encadeamento, e assim por diante.
- O último elemento da lista aponta para NULL, sinalizando que não existe um próximo elemento.

# Declaração de uma lista encadeada

```
typedef struct lista {  
    int info;  
    struct lista* prox;  
} TLista;  
typedef TLista *PLista;
```

# Declaração de uma lista encadeada

```
typedef struct lista {  
    int info;  
    struct lista* prox;  
} TLista;  
  
typedef TLista *PLista;  
  
int  
info prox  
struct lista *
```



# Inicialização de lista encadeada

```
PLista inicializa_lista()  
{  
    return NULL;  
}
```

# Manipulação de Lista Encadeada

- Para cada elemento inserido na lista, devemos alocar dinamicamente a memória necessária para armazenar o elemento e encadeá-lo na lista existente.
- A função de inserção mais simples insere o novo elemento no início da lista.
- O ponteiro que representa a lista deve ter seu valor atualizado, pois a lista deve passar a ser representada pelo ponteiro para o novo primeiro elemento.

# Manipulação de Lista Encadeada

- Inserção
- Busca
- Remoção de um nó na lista
- Impressão de toda a lista
- Libera os espaços alocados para a lista.

## Inserir um elemento na lista (Inicio)

```
void insereInicio (int x, PLista p){  
  
    PLista nova;  
    nova = (TLista *) malloc sizeof(TLista));  
    nova->info = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

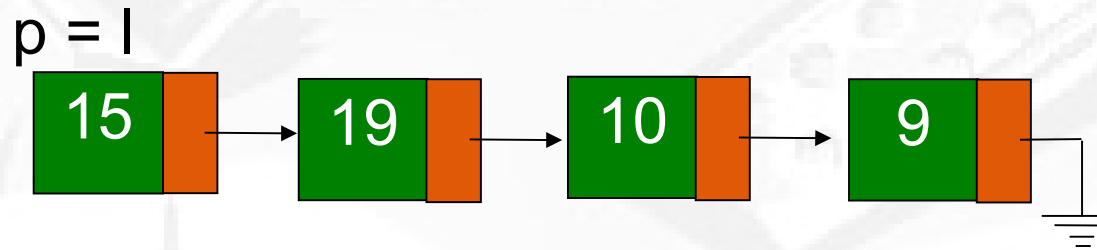
# Procurando um elemento na lista

```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */
}
```

# Procurando um elemento na lista

```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */
}
```

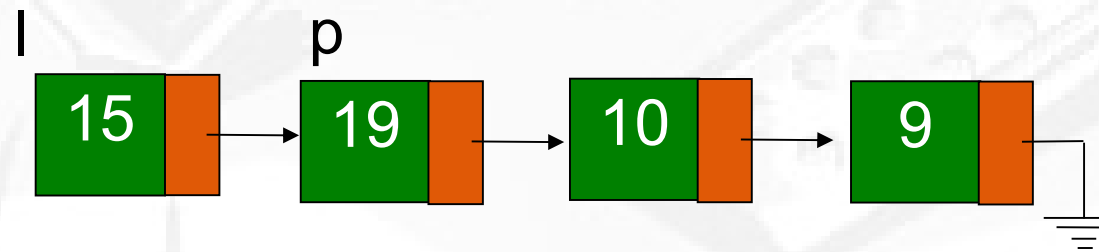
Busca o elemento 9



# Procurando um elemento na lista

```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */
}
```

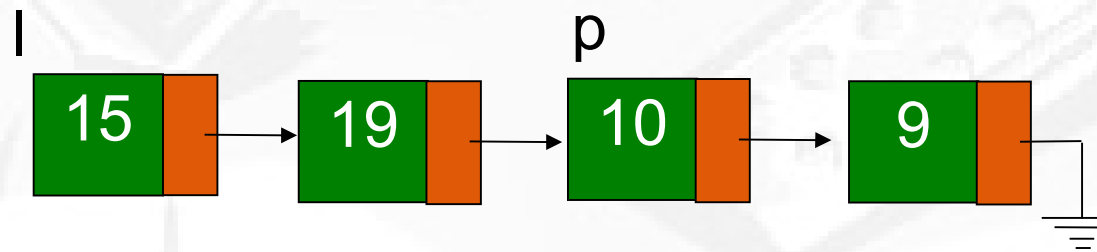
Busca o elemento 9



# Procurando um elemento na lista

```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */
}
```

Busca o elemento 9

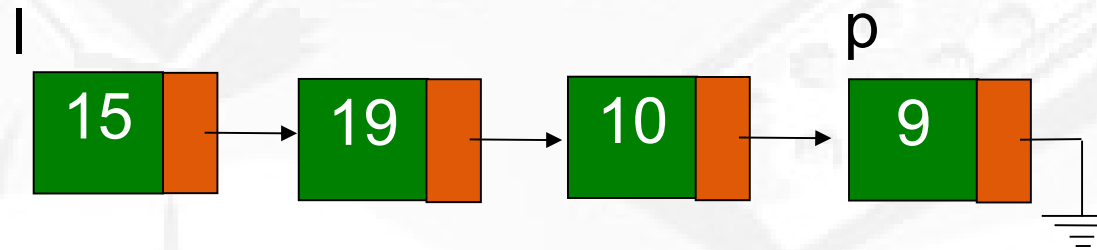




# Procurando um elemento na lista

```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */
}
```

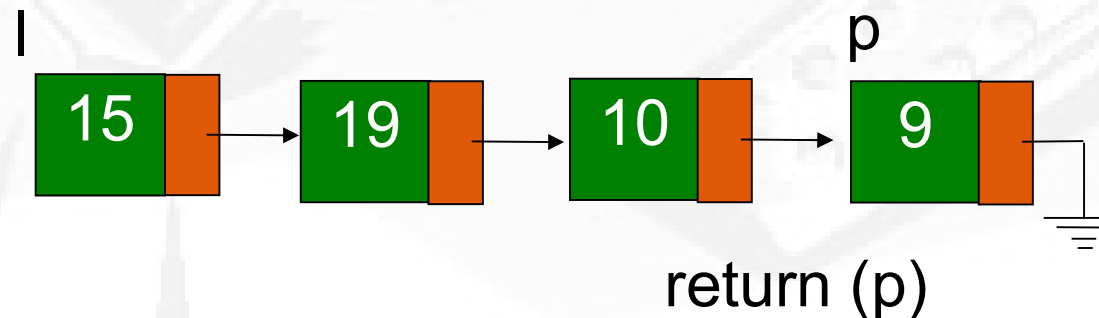
Busca o elemento 9



# Procurando um elemento na lista

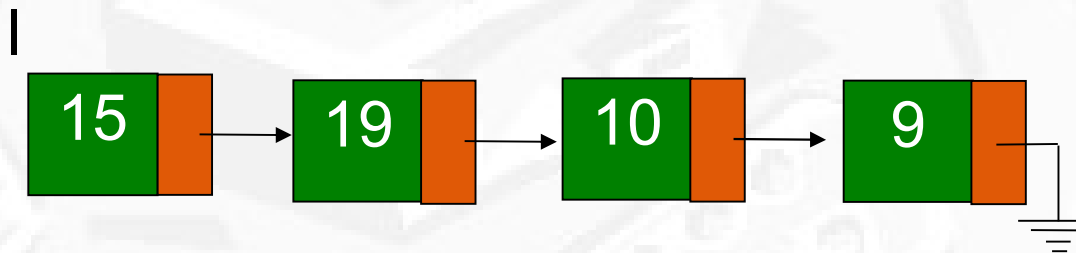
```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */
}
```

Busca o elemento 9



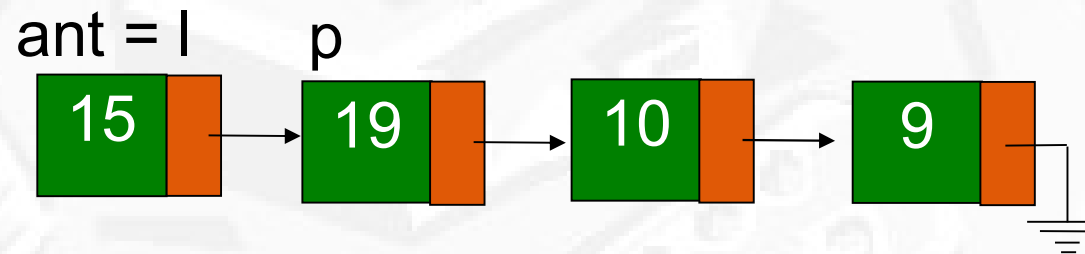
# Retirando da Lista

Retira o elemento 9



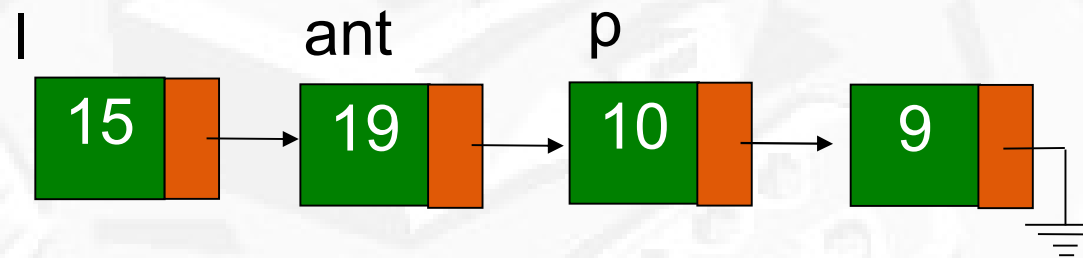
# Retirando da Lista

Retira o elemento 9



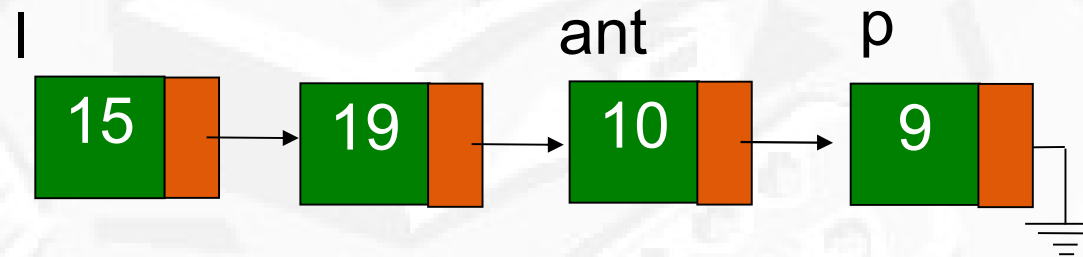
# Retirando da Lista

Retira o elemento 9



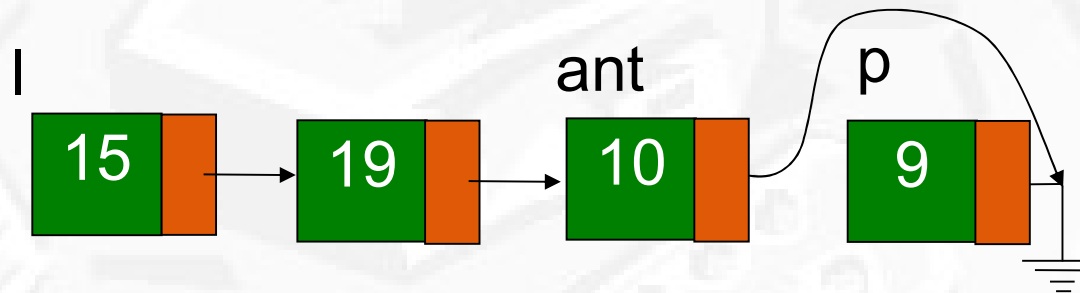
# Retirando da Lista

Retira o elemento 9



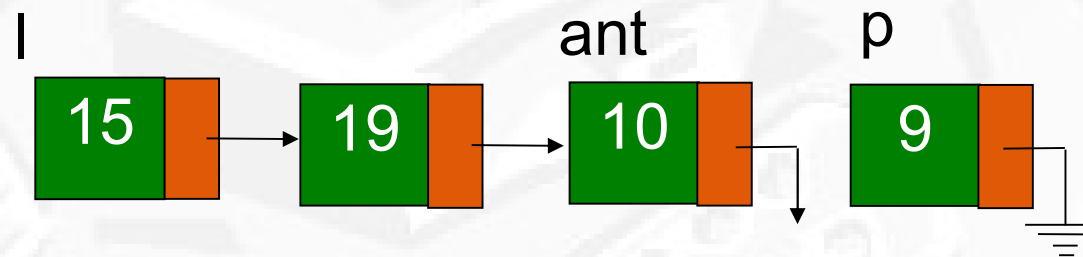
# Retirando da Lista

Retira o elemento 9



# Retirando da Lista

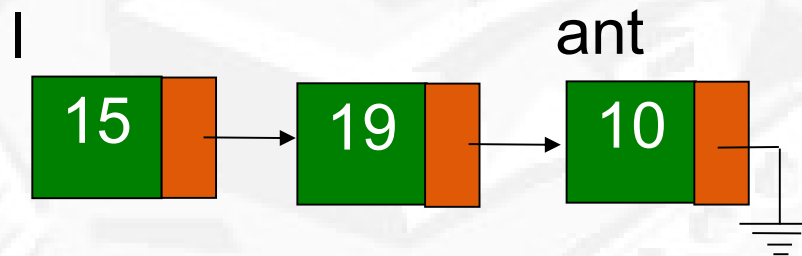
Retira o elemento 9





# Retirando da Lista

Retira o elemento 9



# Retirando da Lista

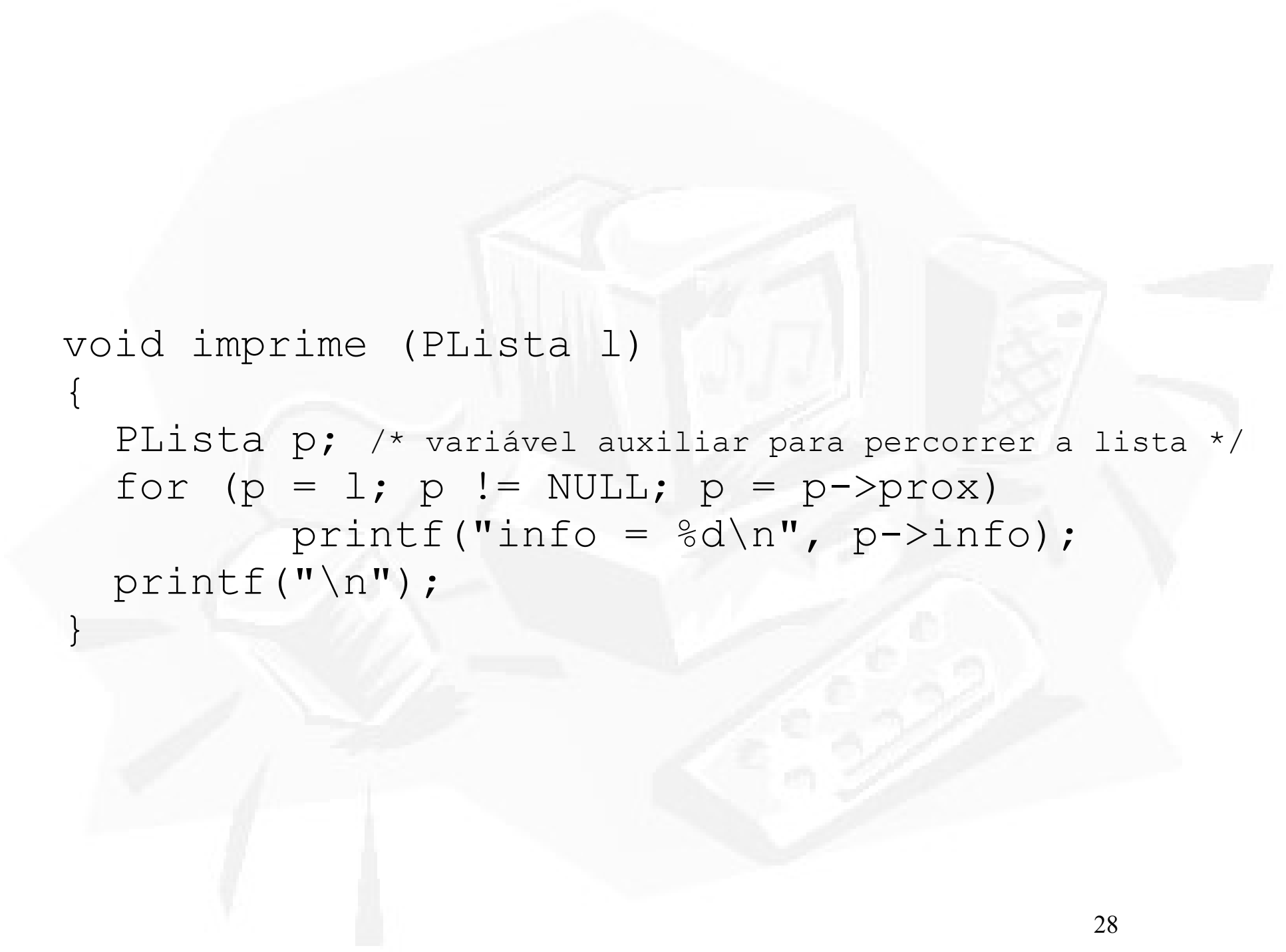
```
PLista retira (PLista l, int v){
    PLista ant = NULL; /* ponteiro para elemento anterior */
    PLista p; /* ponteiro para percorrer a lista*/
    /* procura elemento na lista, guardando anterior */
    for (p=l;p!=NULL && p->info!=v; p = p->prox)
        ant = p;

    /* verifica se achou elemento */
    if (p == NULL)
        return l; /* não achou: retorna lista original */

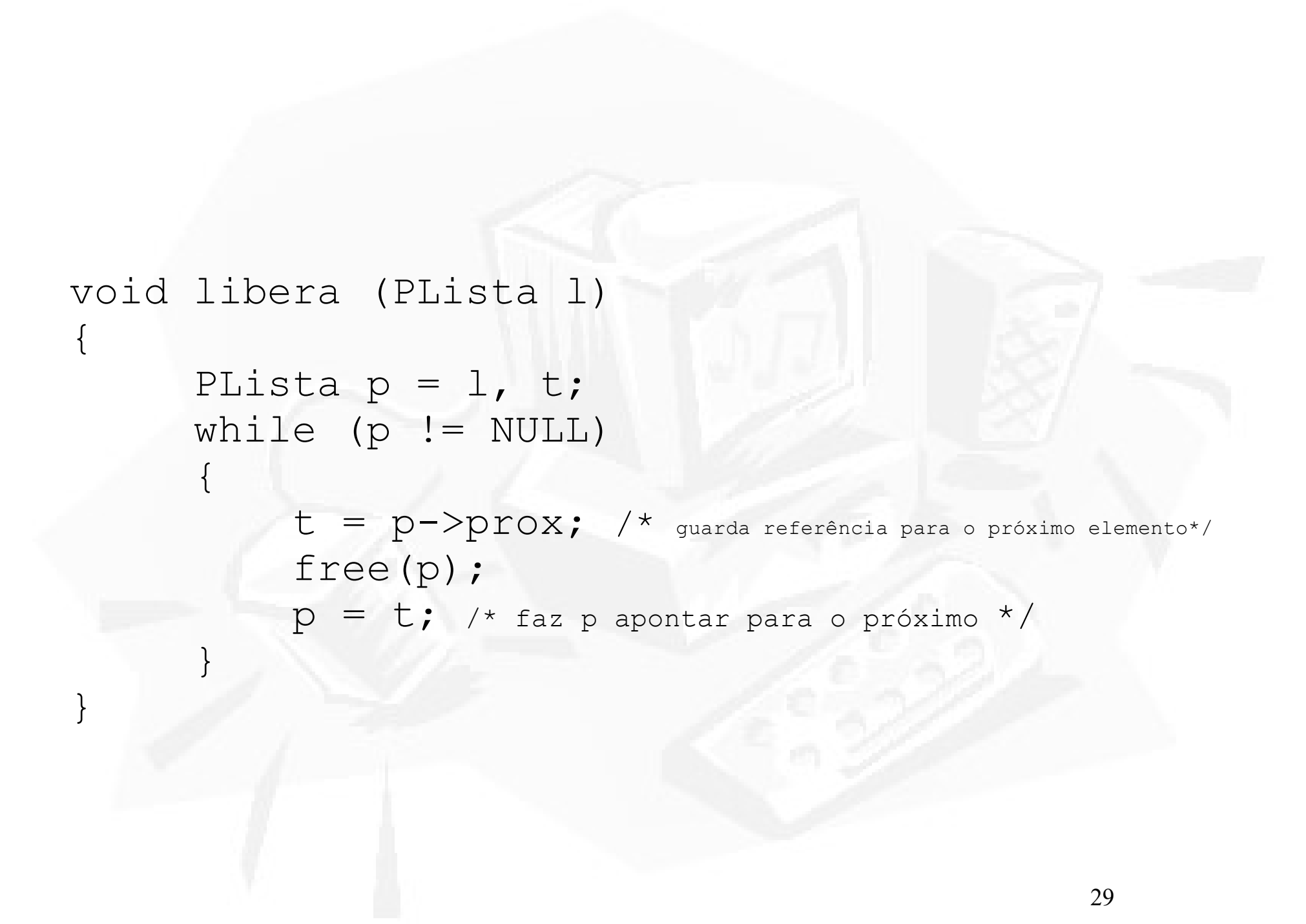
    /* retira elemento */
    if (ant == NULL)
        /* retira elemento do inicio */
        l = p->prox;
    else
        /* retira elemento do meio da lista */
        ant->prox = p->prox;
    free(p);
    return l;
}
```

## Exercício

- Faça a função que imprima toda a lista e a função que libera os espaços alocados.



```
void imprime (PLista l)
{
    PLista p; /* variável auxiliar para percorrer a lista */
    for (p = l; p != NULL; p = p->prox)
        printf("info = %d\n", p->info);
    printf("\n");
}
```



```
void libera (PLista l)
{
    PLista p = l, t;
    while (p != NULL)
    {
        t = p->prox; /* guarda referência para o próximo elemento*/
        free(p);
        p = t; /* faz p apontar para o próximo */
    }
}
```

# Exercícios

- Faça um programa que insira os elementos em ordem crescente.
- Ele deve permitir também a retirada de elementos, conservando a lista ordenada.
- O programa deve ter também as opções de imprimir e de buscar um elemento.

```

PLista Insere_ord (PLista l, int dado)
{
    PLista novo; //novo elemento
    PLista ant = NULL; //ponteiro auxiliar para a posição anterior
    PLista ptaux = l; //ponteiro auxiliar para percorrer a lista

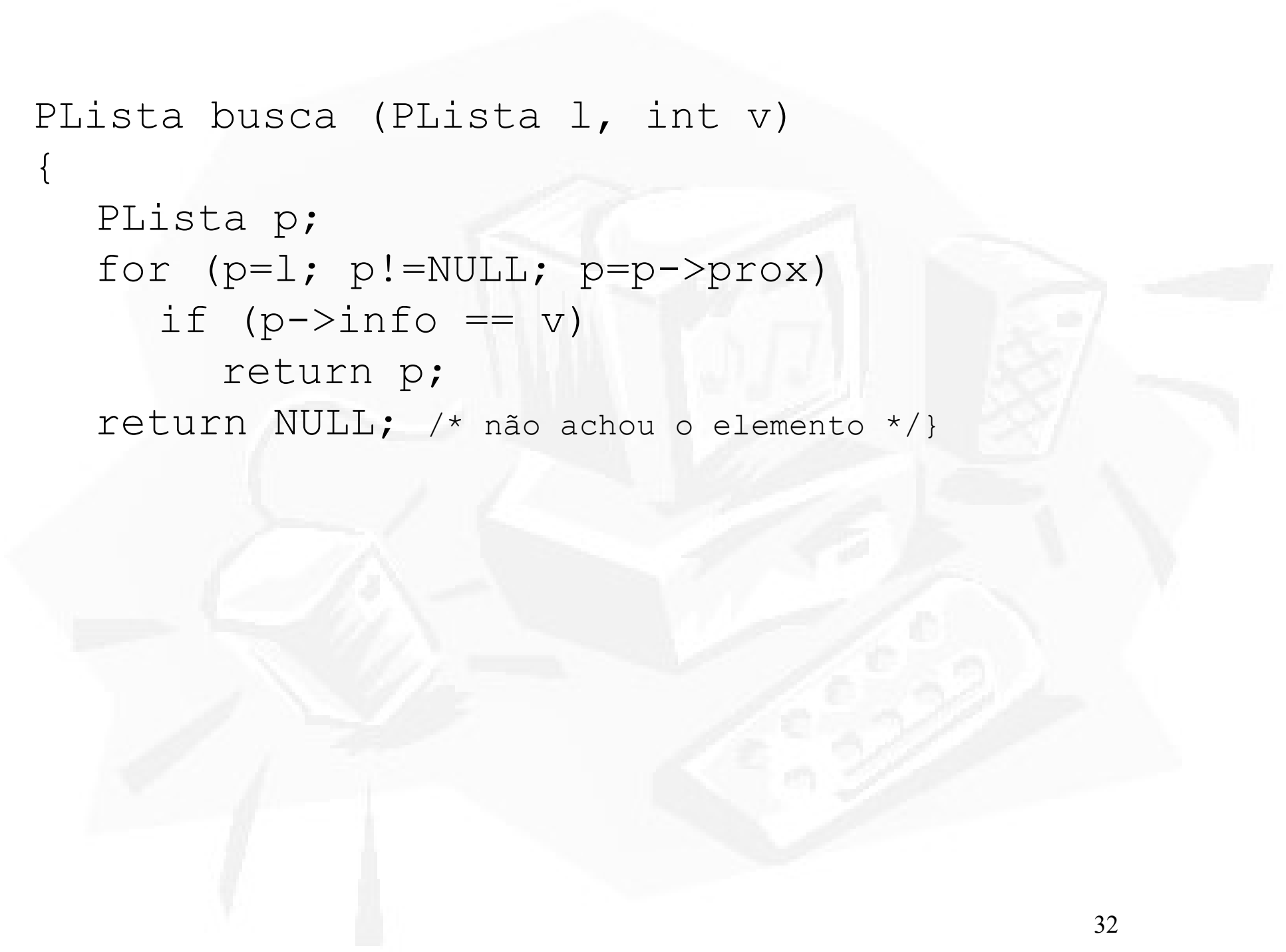
    /*aloca um novo nodo */
    novo = (TLista*) malloc(sizeof(TLista));

    /*insere a informação no novo nodo*/
    novo->info = dado;

    /*procurando a posição de inserção*/
    while ((ptaux!=NULL) && (ptaux->info)<dado)
    {
        ant = ptaux;
        ptaux = ptaux->prox;
    }

    /*encadeia o elemento*/
    if (ant == NULL) /*o anterior não existe, será inserido na 1ª posição*/
    {
        novo->prox = l;
        l = novo;
    }
    else /*elemento inserido no meio da lista*/
    {
        novo->prox = ant->prox;
        ant->prox = novo;
    }
    return l;
}

```



```
PLista busca (PLista l, int v)
{
    PLista p;
    for (p=l; p!=NULL; p=p->prox)
        if (p->info == v)
            return p;
    return NULL; /* não achou o elemento */}
```



```
PLista retira (PLista l, int v)
{
    PLista ant = NULL; /* ponteiro para elemento anterior */
    PLista p; /* ponteiro para percorrer a lista*/

    /* procura elemento na lista, guardando anterior */
    for (p=l;p!=NULL && p->info!=v; p = p->prox)
        ant = p;

    /* verifica se achou elemento */
    if (p == NULL)
        return l; /* não achou: retorna lista original */

    /* retira elemento */
    if (ant == NULL) /* retira elemento do inicio */
        l = p->prox;
    else /* retira elemento do meio da lista */
        ant->prox = p->prox;
    free(p);
    return l;
}
```

```

PLista retira (PLista l, int v)
{
    PLista ant = NULL; /* ponteiro para elemento anterior */
    PLista p; /* ponteiro para percorrer a lista*/

    /* procura elemento na lista, guardando anterior */
    for (p=l;p!=NULL && p->info!=v; p = p->prox)
        ant = p;

    /* verifica se achou elemento */
    if (p == NULL)
        return l; /* não achou: retorna lista original */

    /* retira elemento */
    if (ant == NULL) /* retira elemento do inicio */
        l = p->prox;
    else /* retira elemento do meio da lista */
        ant->prox = p->prox;
    free(p);
    return l;
}

```

# Implementações recursivas

- Uma lista encadeada é representada por:
  - ✓ uma lista vazia; ou
  - ✓ um elemento seguido de uma (sub-)lista.
- Neste caso, o segundo elemento da lista representa o primeiro elemento da sub-lista.

# Função Imprimir recursiva

```
void imprime_rec (PLista l)
{
    if (l==NULL)
        return;
    /* imprime primeiro elemento */
    printf("info: %d\n",l->info);

    /* imprime sub-lista */
    imprime_rec(l->prox);
}
```

# Função Retirar recursiva

```
PLista retira_rec (PLista l, int v)
{
    if (l==NULL)
        return l; //lista vazia: retorna valor original

    // verifica se elemento a ser retirado é o primeiro
    if (l->info == v)
    {
        PLista t = l; // temporário para poder liberar
        l = l->prox;
        free(t);
    }
    else
    {
        /* retira de sub-lista */
        l->prox = retira_rec(l->prox,v);
    }
    return l;
}
```

## Exercício

- Faça uma função recursiva que libera a lista e uma que busque um elemento na lista.

```
void libera_rec (PLista l)
{
    if (l!=NULL)
    {
        libera_rec(l->prox);
        free(l);
    }
}
```

```
PLista busca_r (PLista l, int x)
{
    if (l == NULL)    return NULL;
    if (l->info == x)  return l;
    return busca_r  l->prox, x);
}
```

# Implementação de Fila Dinâmica

- A estrutura de cada nó de uma fila é idêntica à estrutura de cada nó de uma lista:

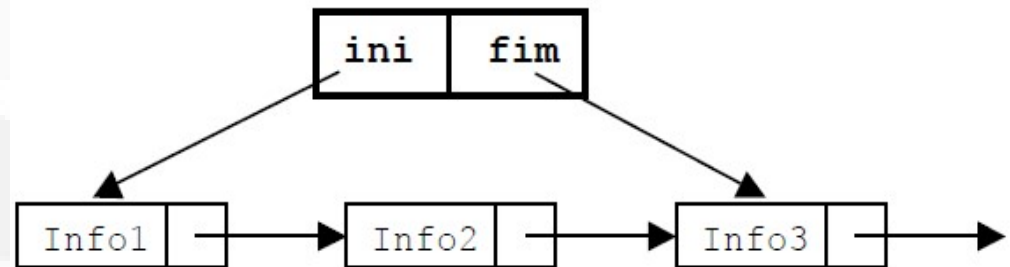
```
typedef struct no {  
    int info;  
    struct no* prox;  
} TNo;  
typedef TNo *PNo;
```



## Estrutura de uma Fila

- No entanto, a estrutura da fila agrupa ponteiros para o início e o fim da fila:

```
typedef struct fila {  
    PNo ini;  
    PNo fim;  
} Tfila;  
typedef Tfila *Pfila;
```



## Inicialização da Fila

- A função `cria` aloca a estrutura da fila e inicializa a lista como sendo vazia

```
PFile cria() {  
    PFile f = (PFile) malloc(sizeof(TFile));  
    f->ini = f->fim = NULL;  
    return (f);  
}
```

# Exercício

- Escreva as funções de inserção, remoção, impressão e liberação de uma fila encadeada. Lembre que a inserção deve sempre ser no fim da fila e a remoção, no início.

```

/* insere no fim */
PFile insere (PFile f, int v) {
    PNo novo = (PNo) malloc(sizeof(TNo));
    novo->info = v;
    novo->prox = NULL;
    if (f->fim != NULL) /* verifica se lista não estava vazia */
        f->fim->prox = novo;
    else f->ini = novo; /* fila vazia */
    f->fim = novo;
    if (f->ini == NULL) f->ini = novo;
    return (f);
}

/* retira do início */
PFile retira (PFile f, int *v) {
    PNo p;
    if (f->ini==NULL) /* fila vazia */
        printf("\nFila vazia!!\n");
    else{
        *v = f->ini->info;
        p = f->ini;
        if (f->ini == f->fim) { /* só tem um nó na fila */
            f->ini = f->fim = NULL;
        }
        else
            f->ini = p->prox; /* ou f->ini = f->ini->prox; */
        free(p);
    }
    return f;
}

```

```
/* Imprime fila (sempre do início para o fim) */
PNo p;
    if (f->ini==NULL) /* fila vazia */
        printf("\nFila vazia!!\n");
    else {
        for (p = f->ini; p!=NULL; p = p->prox)
            printf("%d ",p->info);
    }

/* libera: quase igual à que libera lista */
void libera (PFila f) {
    PNo p;
    for (p = f->ini; p!=NULL; p = f->ini) {
        f->ini = p->prox; // ou f->ini = f->ini->prox;
        free(p);
    }
    free(f);
}
```

# Exercícios

- Faça uma função que receba 3 filas (f, f\_pares e f\_impares) e separe todos os valores guardados em f de tal forma que os valores pares sejam movidos para f\_pares e os impares, para f\_impares. No final, f deve estar vazia. Considere que f\_pares e f\_impares ainda não existem.

```
/* Separa os valores pares dos ímpares*/  
  
void separa_fila(Pfila f, Pfila f_pares, Pfila  
f_impares){  
    Pfila p;  
    int val;  
    f_pares = cria();  
    f_impares = cria();  
    p = retira(f, &val);  
    while(p!=NULL){  
        if(val%2==0)  
            f_pares = insere(f_pares, val);  
        else  
            f_impares = insere(f_impares, val);  
        r = retira(f, &val);  
    }  
}
```







**FIM**