Estruturas de Dados

Engenharia Informática

1º Ano - 2º Semestre

Francisco Morgado
Escola Superior de Tecnologia e Gestão de Viseu

3. ESTRUTURAS DINÂMICAS

3.1 Stacks (Pilhas)

3.2 Filas de espera

3.3 Listas ligadas ordenadas

3.4 Listas bi-ligadas ordenadas

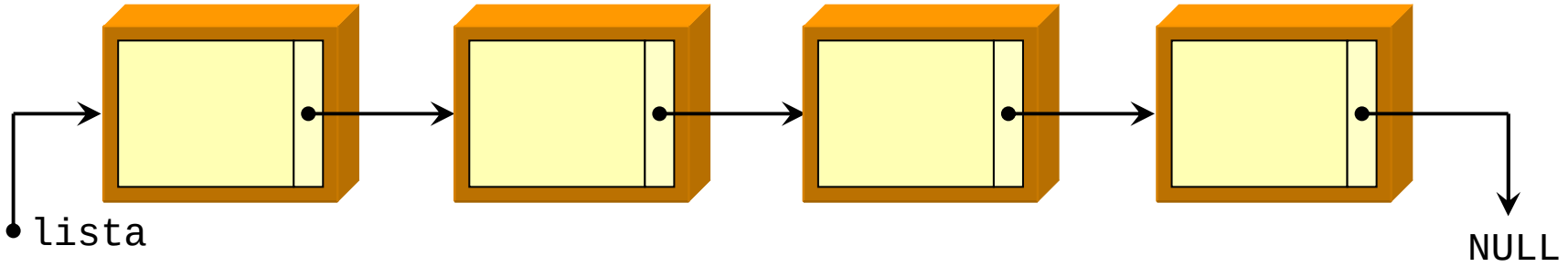
3.5 Hashing

3.6 Árvores binárias

3.3 Listas ligadas ordenadas

- *Conceito*
- *Inserção de elementos*
- *Remoção de elementos*
- *Listagem de todos os elementos*
- *Aplicação a uma lista com elementos (relativamente) genéricos*

Listas Ligadas Ordenadas - Conceito




```
typedef struct
 {
       char nome[80];
       int numero; /*chave*/
       float mediaCurso;
}Informacao;
```

```
typedef struct nodo
 {
   Informacao dados;
   struct nodo* seguinte;
 }Nodo;
```

```
typedef Nodo* Lista;
```

```
// criar lista vazia
Lista lista = NULL ;
```

- **Operações na Lista**
  - Inserir elemento na ordem correta
  - Retirar qualquer elemento
  - Listar elementos (ordenadamente)

Enumerações

- **Lista numerada de constantes inteiras**
  - valores booleanos
  - meses do ano
  - avaliação qualitativa

```
enum boolean { false, true}; /* false = 0, true = 1*/
typedef enum boolean bool;
```

```
enum mesE {Janeiro = 1, Fevereiro, ..., Dezembro};/* 1-12 */
typedef enum mesE Mes;
```

```
enum aval{MuitoMau = -2, Mau = -1, Nulo = 0, Bom = 1,
MuitoBom = 2};
typedef enum aval Avaliacao;
```

O compilador pode não garantir a restrição aos valores válidos.

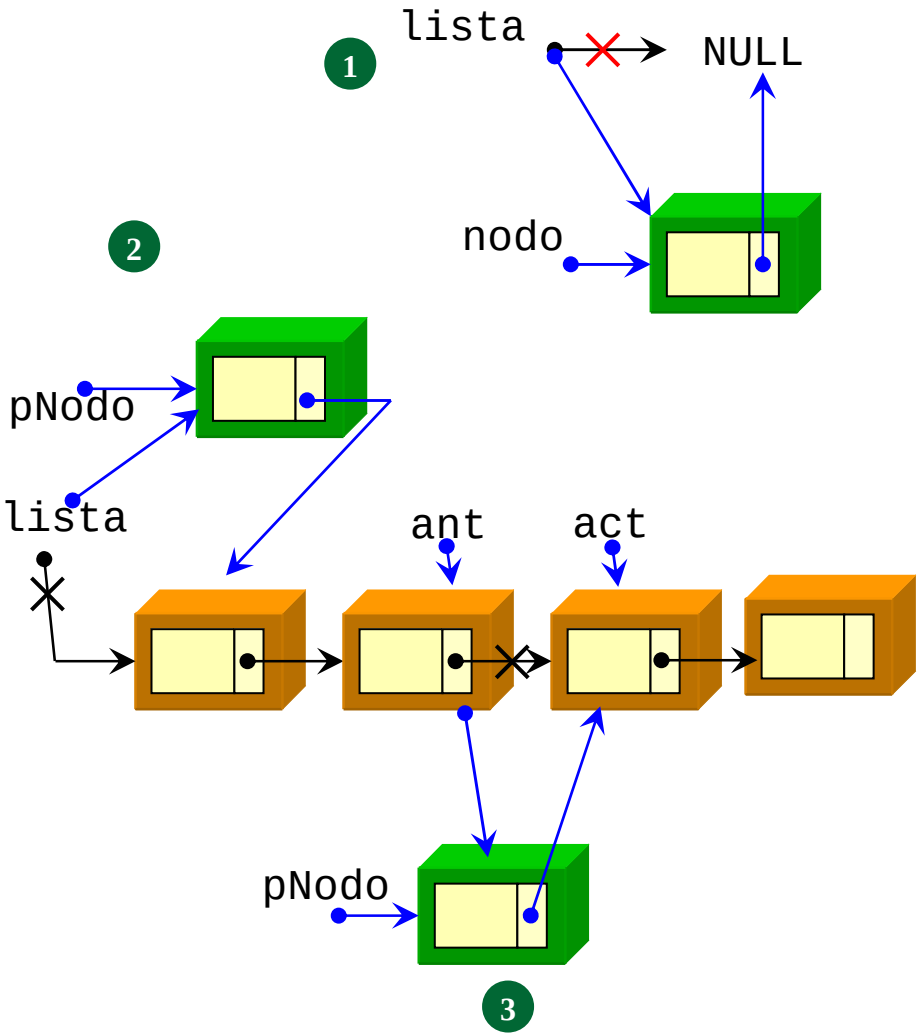
Listas Ligadas Ordenadas - Inserção de elementos

```
void inserir(Informacao elemento, Lista* lista)
{
 bool procura = true; Nodo *anterior, *atual;
 Nodo* pNodo = (Nodo*)malloc(sizeof(Nodo));
 pNodo->dados = elemento;
 if (*lista == NULL) // lista estava vazia
  {  pNodo->seguinte = NULL; *lista = pNodo;}
 else    // lista já continha elemento(s)
  {
   anterior = atual = *lista;
   while(procura)
     if(atual == NULL)
       procura = false;
     else if(atual->dados.numero >
                    pNodo->dados.numero)
              procura = false;
            else
              { anterior = atual;
                atual = atual->seguinte; }
   if(atual == *lista) // insere no início
    {pNodo->seguinte = *lista; *lista = pNodo;}
   else  // insere no meio ou fim
    { pNodo->seguinte = atual;
      anterior->seguinte = pNodo; }
  }
}
```

Ordem crescente

Chamada

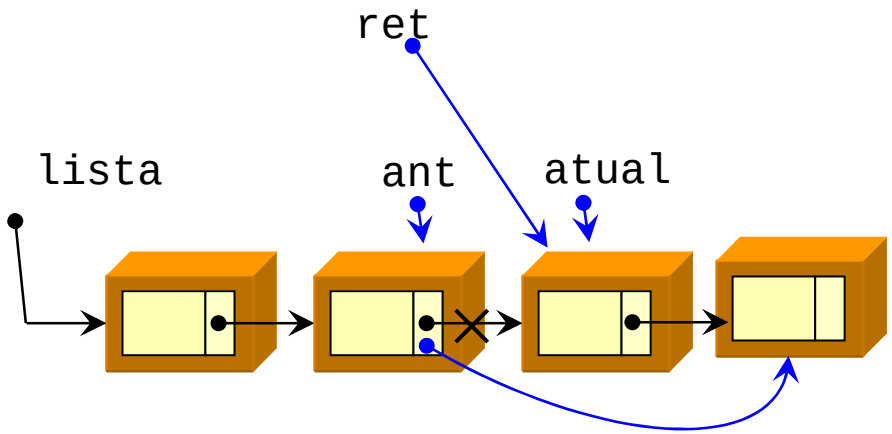
inserir(elemento, &lista);



```
Nodo* remover(int chave, Lista* lista)
{
 bool procura=true; Nodo* ret, *ant, *atual;
 if (*lista == NULL)
   ret = NULL;
 else
  {
   ant = atual = *lista;
   while(procura)
     if(atual == NULL)
       procura = false;
     else if(atual->dados.numero == chave)
       procura = false;
     else
      {
       ant = atual; atual = atual->seguinte;
      }
   if(atual != NULL)
    {
     ret = atual;
     if(*lista == atual)
       *lista = atual->seguinte;
     else
      ant->seguinte = atual->seguinte;
    }
   else
     ret = NULL;
  }
 return(ret);
}
```

Chama-se fazendo

pNodo = remover(nAl, &lista);



```
void listar(Lista lista)
{
 int i=1;
 while (lista != NULL)
    {
     printf("\n %dº Elemento:", i++);
     mostrar(lista->dados);
     lista = lista->seguinte;
    }
}
```

Chama-se usando

```
listar(lista);
```

Aplicação a uma lista com elementos (relativamente) genéricos

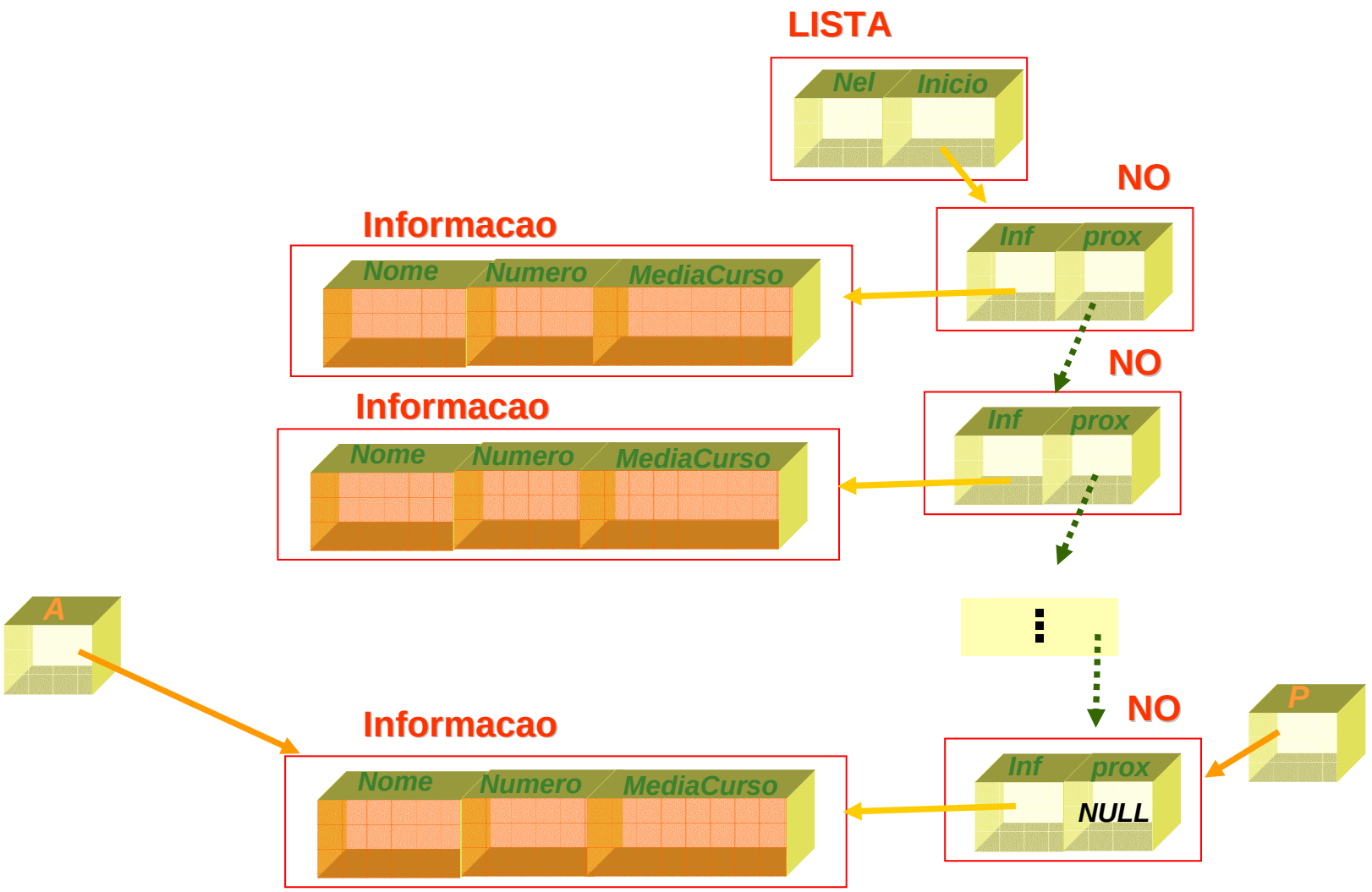
Estrutura dos elementos

```
typedef struct
{
      char nome[MAX_NOME];
      int  numero;
      float mediaCurso;
} Aluno;
```

```
#define MAX_NOME 80 // Tamanho máximo para o nome
```

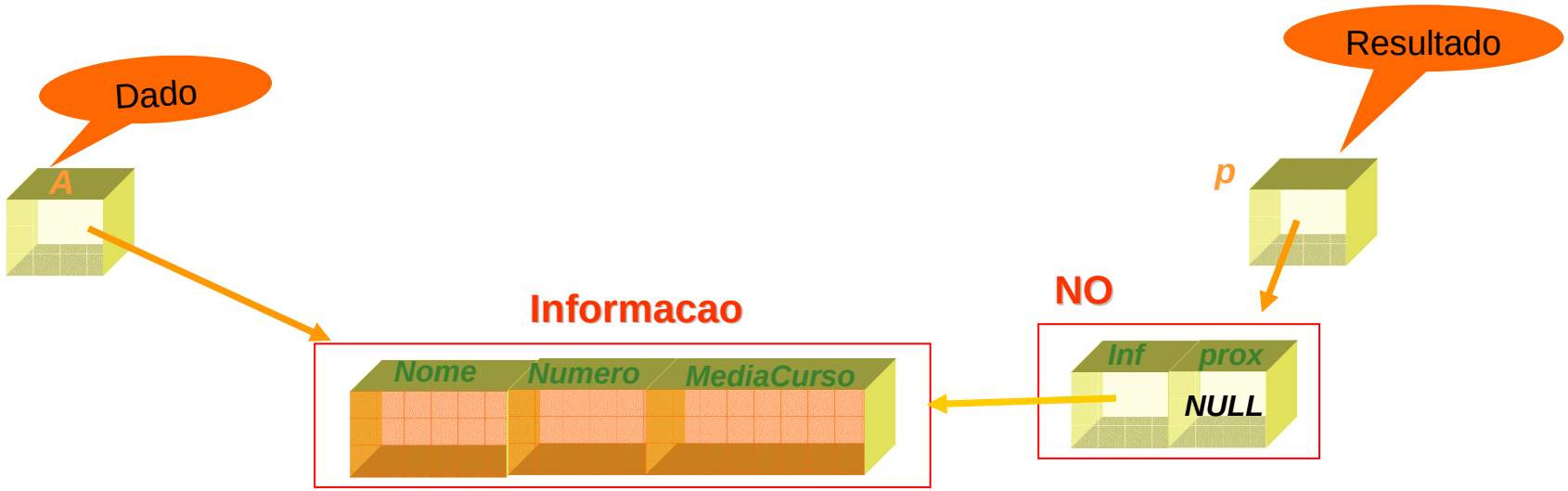
Estrutura da Lista

```
typedef Aluno Informacao;
typedef struct no
{
      Informacao* inf;
      struct no* prox;
}No, *NO;
typedef struct lista
{
  int nElementos;//Permite a todo o momento conhecer o nº de Nós da Lista
  NO inicio;     //Início da sequência dos Nós;
}Lista, *LISTA;
```

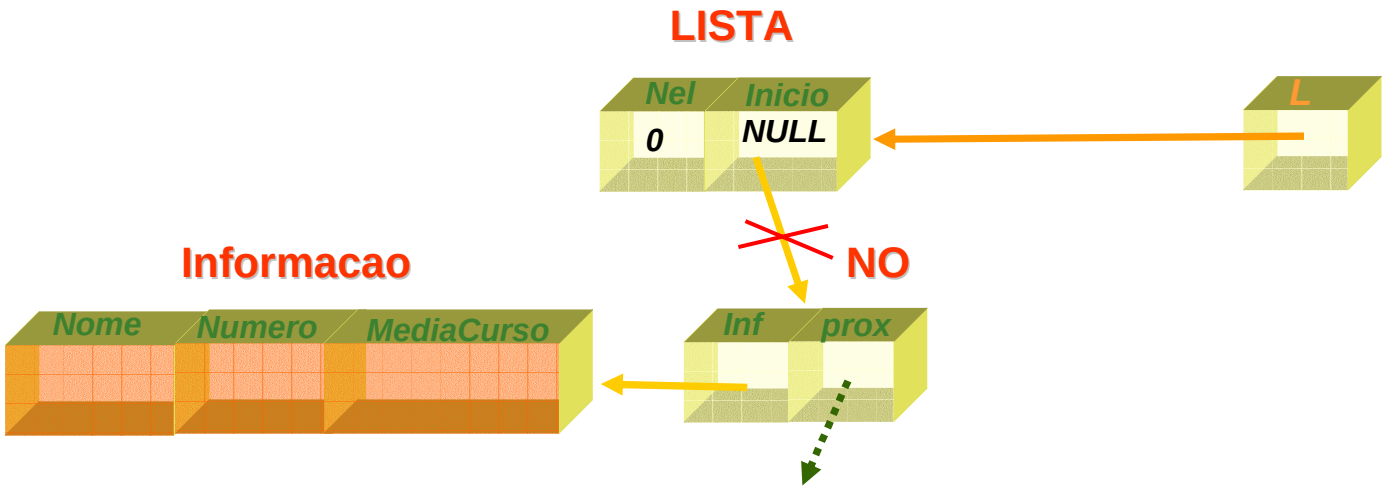

LISTA
Nel
Inicio
NO
Inf
prox
Informacao
Nome
Numero
MediaCurso
NO
Inf
prox
Informacao
Nome
Numero
MediaCurso
A
NO
P
Inf
prox
NULL
Informacao
Nome
Numero
MediaCurso

Funções de Apoio

```
NO criarNo (Informacao* pElemento)
{
      NO p = (NO)malloc(sizeof(No));
      p->inf = pElemento;
      p->prox = NULL;
      return p;
}
```



Criar uma lista

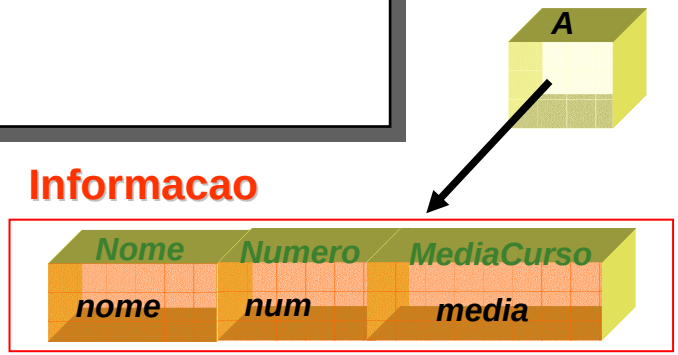



```
Lista* criarLista()
{
      LISTA lista; // ou Lista *lista;
      lista = (LISTA)malloc(sizeof(Lista));
      lista->inicio = NULL;
      lista->nElementos = 0;
      return lista;
}
```

```
Informacao *criarUmElemento(char *nome, int num, float media)
{
      Informacao *A = (Informacao *)malloc(sizeof(Informacao));
      //A fica a apontar para o bloco criado, do tipo Informacao
      strcpy(A->nome, nome);
      A->numero = num;
      A->mediaCurso = media;
      return A;
}
```

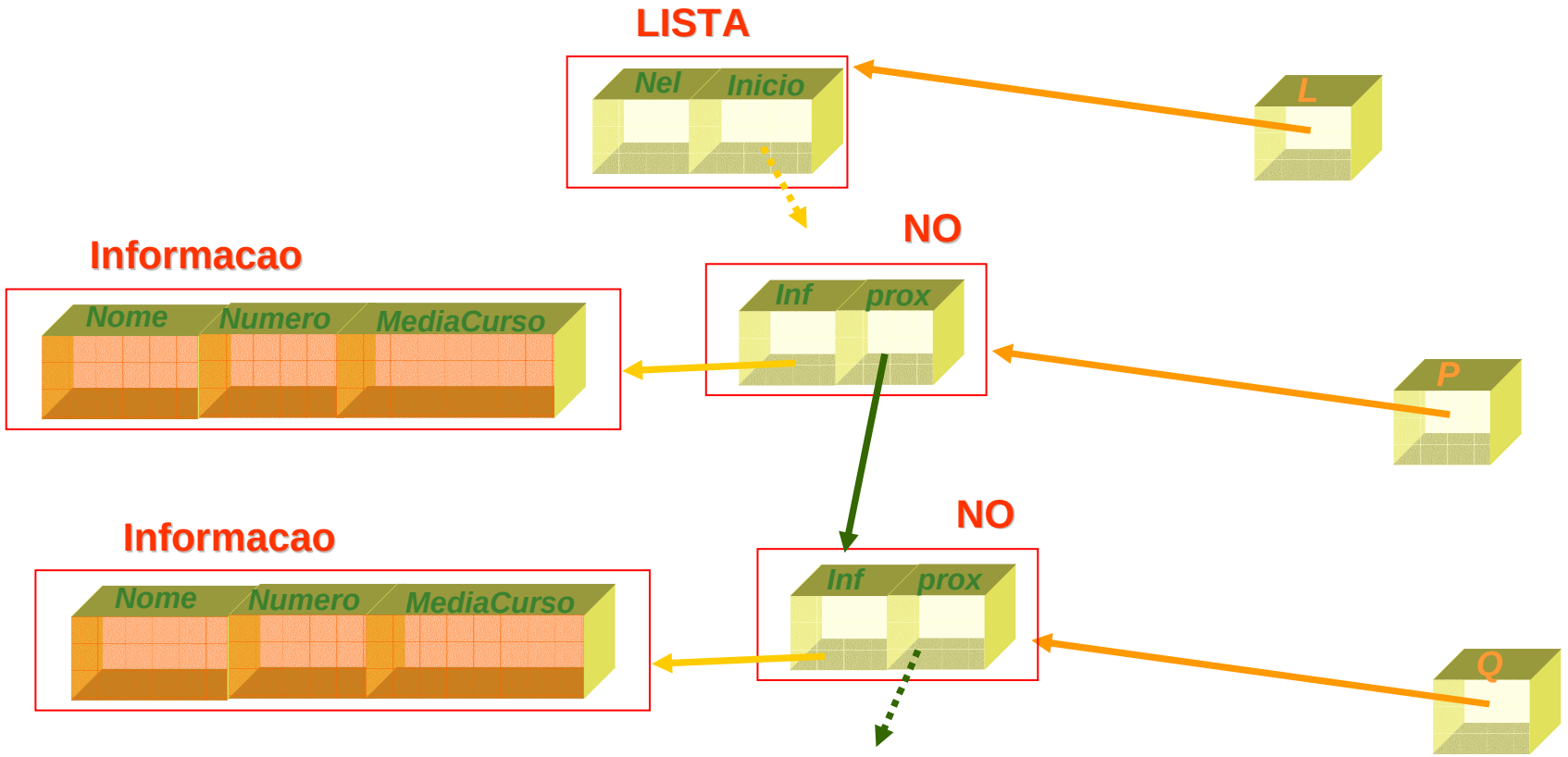
EXEMPLO DE CHAMADA:

```
a = criarUmElemento("Manuel",1234,(float)14);
```




```
void mostrarInformacao(Informacao * a)
{
      printf("Nome do Aluno= %s\n",   a->nome);
      printf("Numero do Aluno= %d\n", a->numero);
      printf("Media do Curso= %f\n",  a->mediaCurso);
}
```

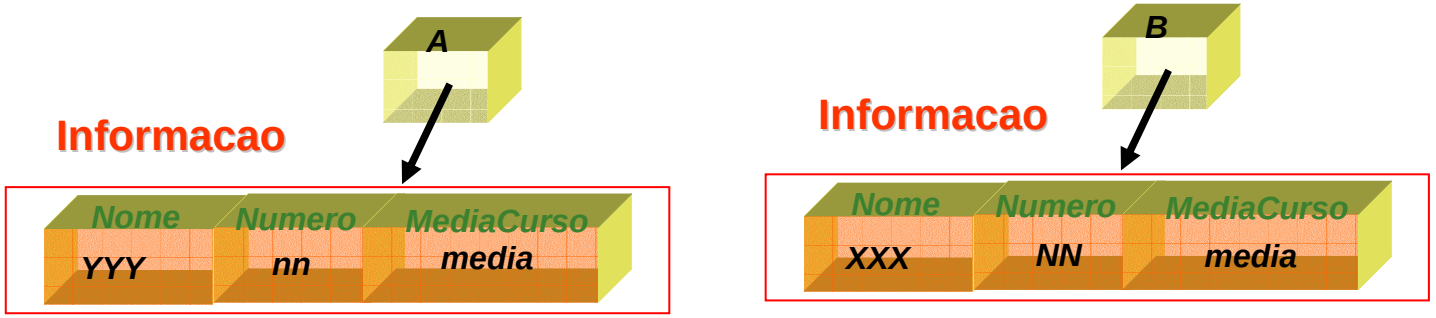
```
void actualizarSeguinte(NO p, NO q)
{
   p->prox = q;
}
```



```
void gravarElementoFicheiro(FILE * f, Informacao * a)
{
      fprintf(f, "Nome-do-Aluno= %s\n",   a->nome);
      fprintf(f, "Numero-do-Aluno= %d\n", a->numero);
      fprintf(f, "Media-do-Curso= %f\n",  a->mediaCurso);
}
```

```
Informacao * lerElementoFicheiro(FILE * f)
{
      char nome[MAX_NOME];
      int num;
      float media;
      fscanf(f,"%s", nome);
      fscanf(f,"%d", &num);
      fscanf(f,"%f", &media);
      return criarUmElemento(nome, num, media);
}
```

```
bool elementosIguais(Informacao * a, Informacao * b)
{
      // Vamos verificar se a é igual a b.
      // Poderemos verificar através do nome ou do número
      return a->numero == b->numero; // Através do Numero
      // return strcmp(a->nome, b->nome) == 0; // Através do Nome
}
```



RECORDAR QUE

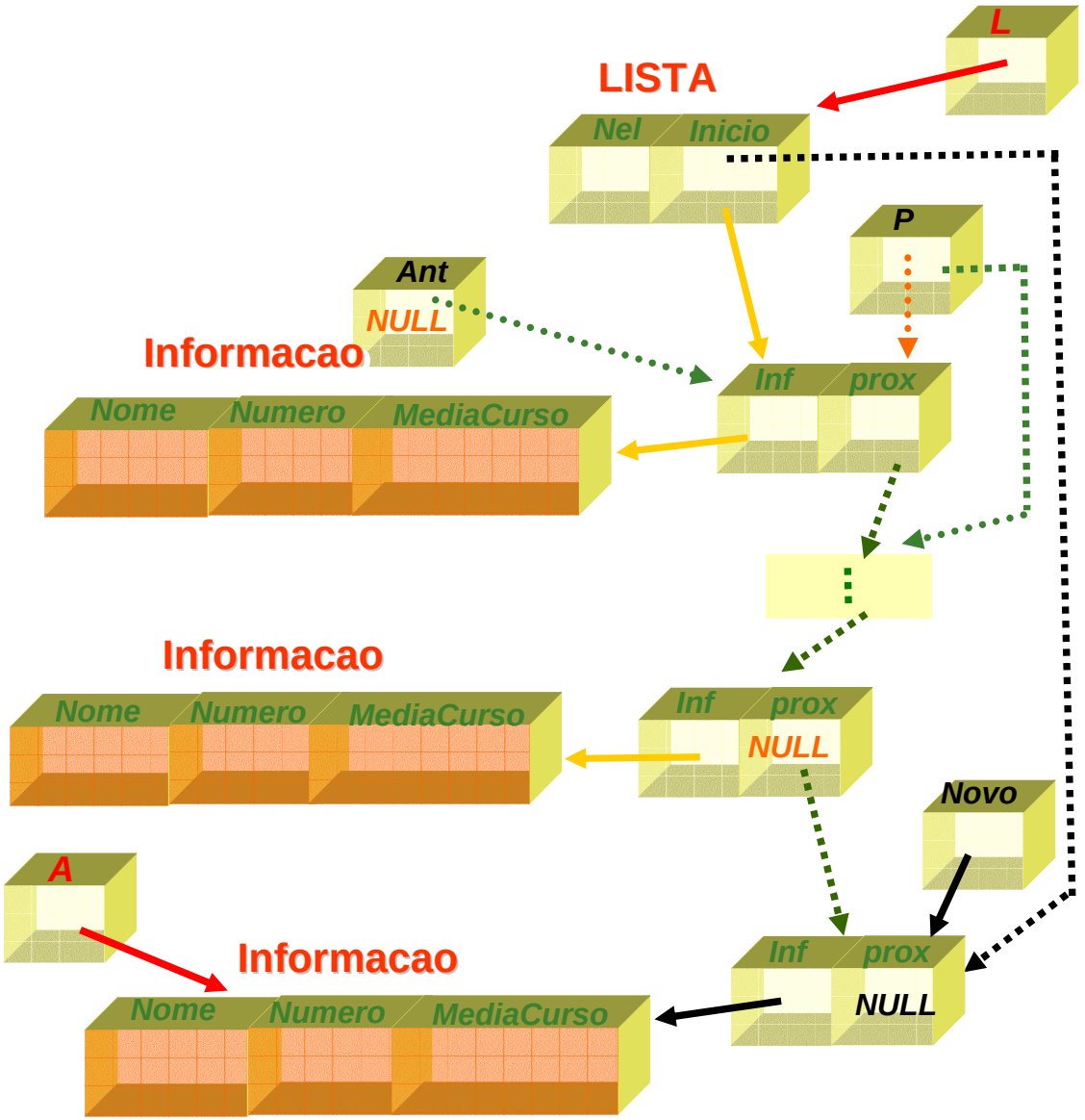
A função `strcmp (char *s1, char *s2)` compara alfabeticamente as *strings* s1 e s2.
Devolve um inteiro:

- <0 se s1 é alfabeticamente menor que s2
- 0 se s1 e s2 são iguais
- >0 se s1 é alfabeticamente maior que s2

Inserir no fim da lista (um Elemento)

Cabeçalho

```
void inserirFimLista(LISTA l, Informacao * a)
```



Função no diapositivo seguinte

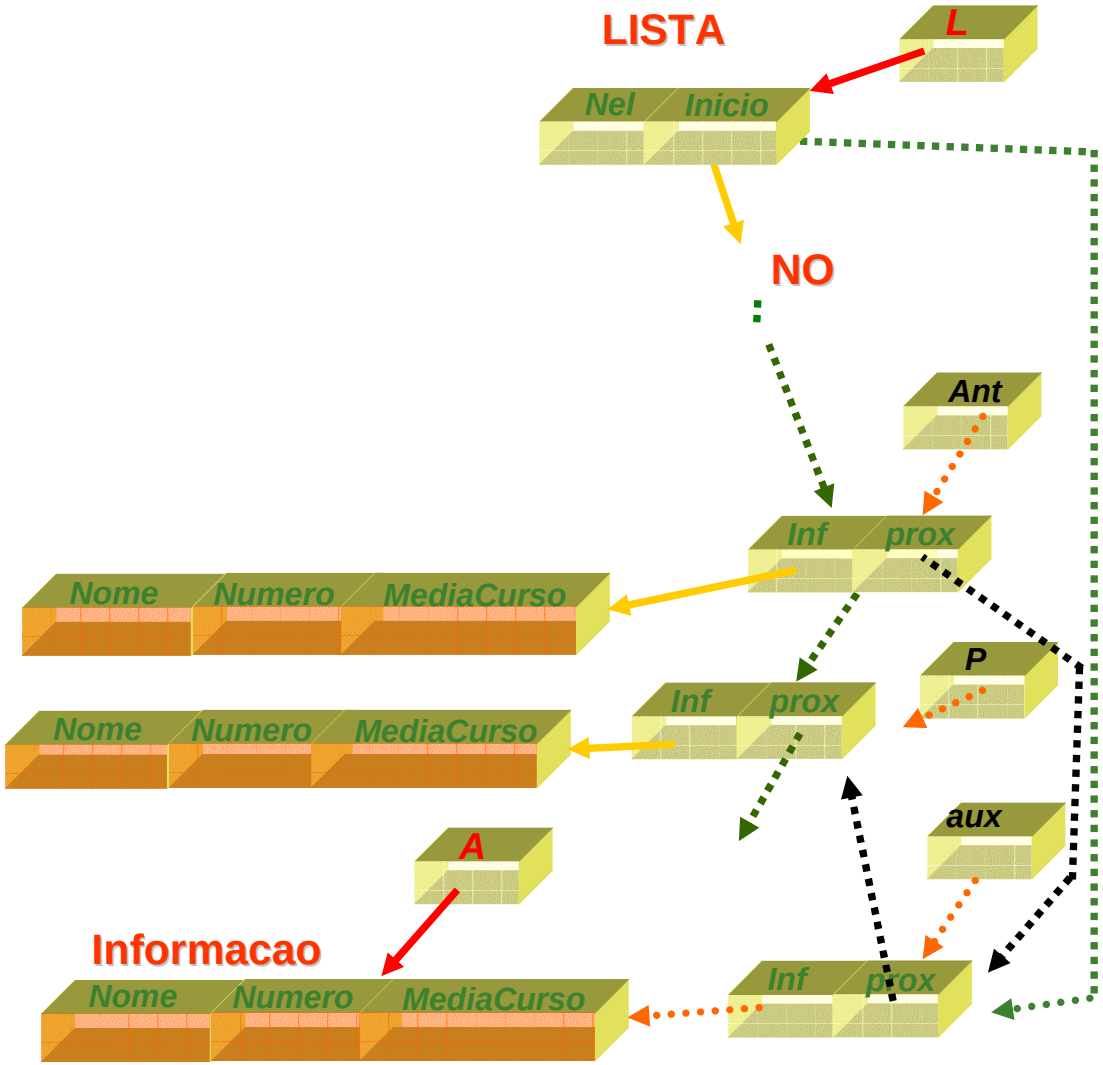
Inserir um elemento no fim da lista - Função

```
void inserirFimLista(LISTA l, Informacao * a)
{
	if (l == NULL) return;
	NO novo = criarNo(a);
	// Temos de ir andando na Lista e quando chegar ao Fim....
	// Temos de ter sempre um ponteiro para o anterior ...
	if (l->inicio == NULL) // Caso em que a Lista não tem nada ...
		l->inicio = novo;
	else // A Lista já tem alguma coisa
	{
		NO p = l->inicio;
		while (p->prox != NULL)
			p = p->prox;
	// Nesta altura p é um ponteiro para o último elemento da lista
		actualizarSeguinte(p, novo);
	}
	l->nElementos++;
}
```

Inserir um elemento, garantindo a ordenação da lista

Cabeçalho

```
void inserirOrd(LISTA l, Informacao * a)
```



Função no diapositivo seguinte

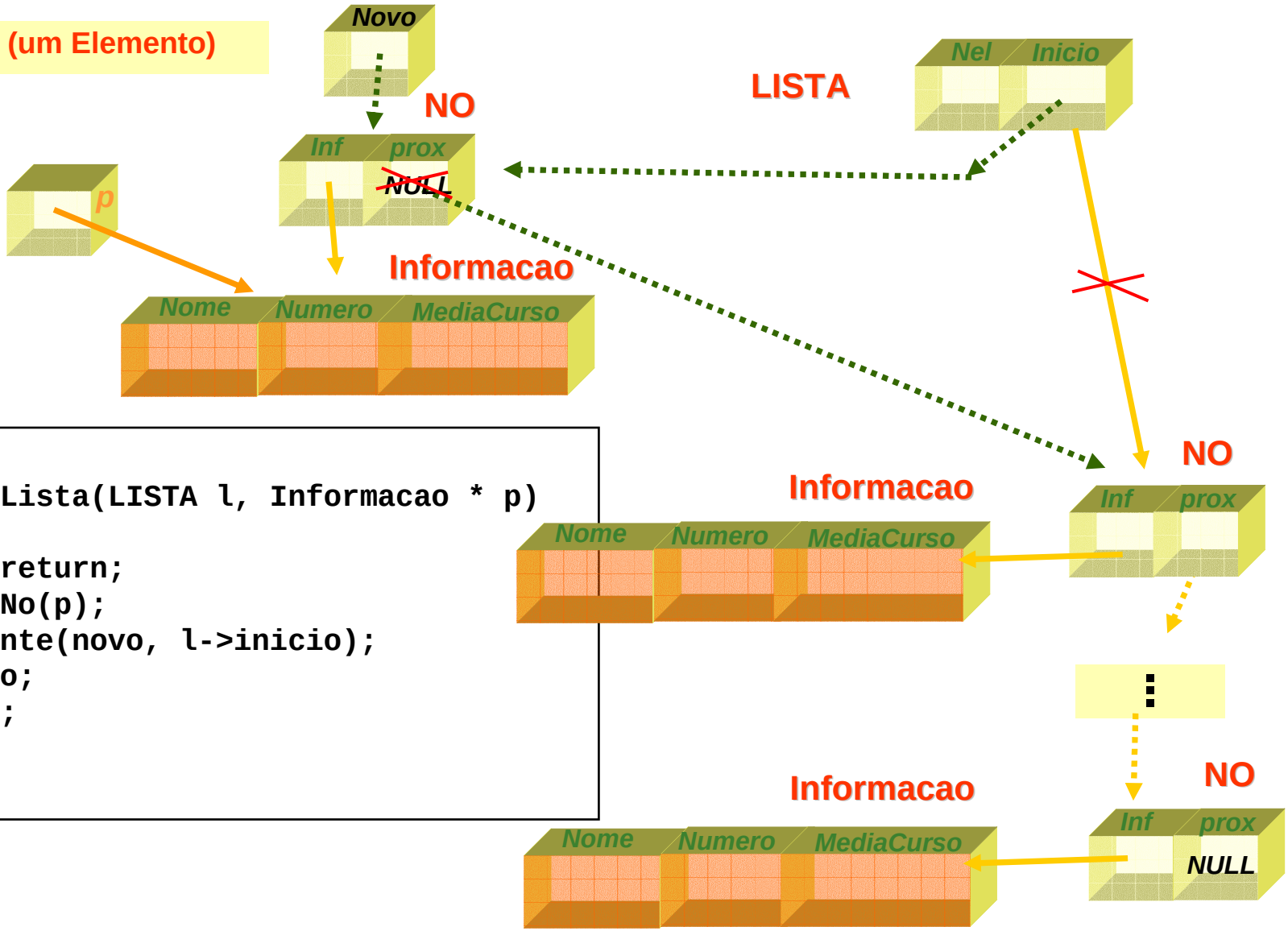
Inserir um elemento, garantindo a ordenação da lista - Função

```
void inserirOrd(LISTA l, Informacao * a)
{
	if (l == NULL) return;
	NO ant = NULL;
	NO p = l->inicio;
	bool parar = false;
	while (p && !parar)
	{
		if (p->inf->mediaCurso > a->mediaCurso)//ordem crescente
			parar = true;
		else
		{
			ant = p;
			p = p->prox;
		}
	}
	NO aux = criarNo(a);
	actualizarSeguinte(aux, p);
	if (ant != NULL) // inserir o Elemento entre ant e p
		actualizarSeguinte(ant, aux);
	else // A inserção será feita no inicio
		l->inicio = aux;
	l->nElementos++;
}
```

Remover um elemento, dado uma chave (nº aluno) - Função

```
NO remover(int chave, LISTA lista)
{
 bool procura=true; NO ret, ant, atual;
 if (lista->inicio == NULL)
   ret = NULL;
 else
  {
   ant = atual = lista->inicio;
   while(procura)
     if(atual == NULL)
       procura = false;
     else if(atual->inf->numero == chave)
       procura = false;
     else
      {    ant = atual; atual = atual->prox;
      }
   if(atual != NULL)
    {
     ret = atual;
          lista->nElementos--;
     if(lista->inicio == atual)
       lista->inicio = atual->prox;
     else
      ant->prox = atual->prox;
    }
   else
     ret = NULL;
  }
 return(ret);
}
```

Inserir no início da lista (um Elemento)




```
void inserirInicioLista(LISTA l, Informacao * p)
{
   if (l == NULL) return;
   NO novo = criarNo(p);
   actualizarSeguinte(novo, l->inicio);
   l->inicio = novo;
   l->nElementos++;
}
```

Função iterativa para verificar se um dado Elemento pertence à lista

```
bool pertenceIterativo(LISTA l, Informacao * a)
{
	if (a == NULL || l == NULL) return false;
	NO p = l->inicio;
	while (p != NULL)
	{
		if (elementosIguais(p->inf, a))
			return true;
		else
			p = p->prox;
	}
	return false;
}
```

Função recursiva para verificar se um dado Elemento pertence à lista

```
bool pertenceRecNo(NO p, Informacao * a)
{
      if (p == NULL)
            return false;
      if (elementosIguais(p->inf, a))
            return true;
      else return pertenceRecNo(p->prox, a);
}

bool pertenceRecursivo(LISTA l, Informacao * a)
{
      if (a == NULL || l == NULL) return false;
      /* bool pert = pertenceRecNo(l->inicio, a);
       return pert;*/
      return pertenceRecNo(l->inicio, a);
}
```

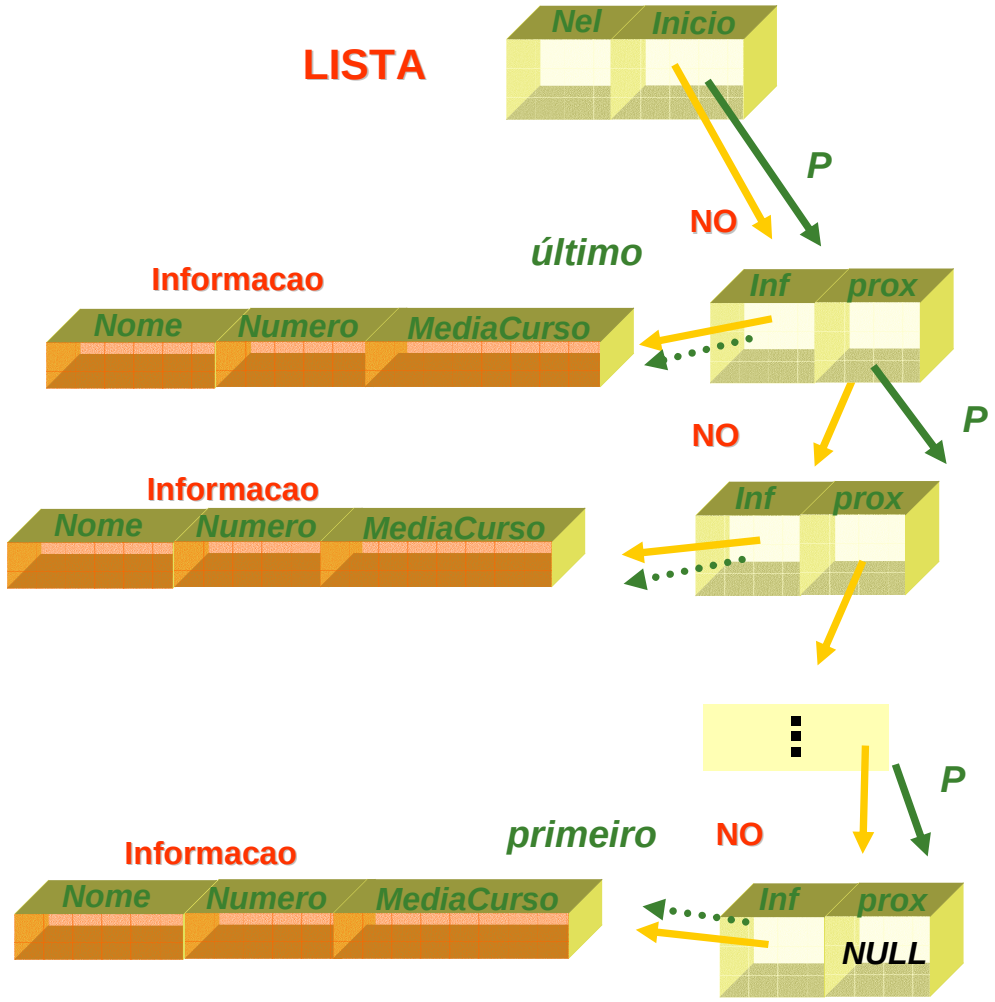
Mostrar todo o conteúdo da lista pela ordem em que se encontram os elementos

```
void listarLista(LISTA l)
{
      if (l == NULL) return;
      NO p = l->inicio;
      while (p != NULL)
      {
            mostrarInformacao (p->inf);

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

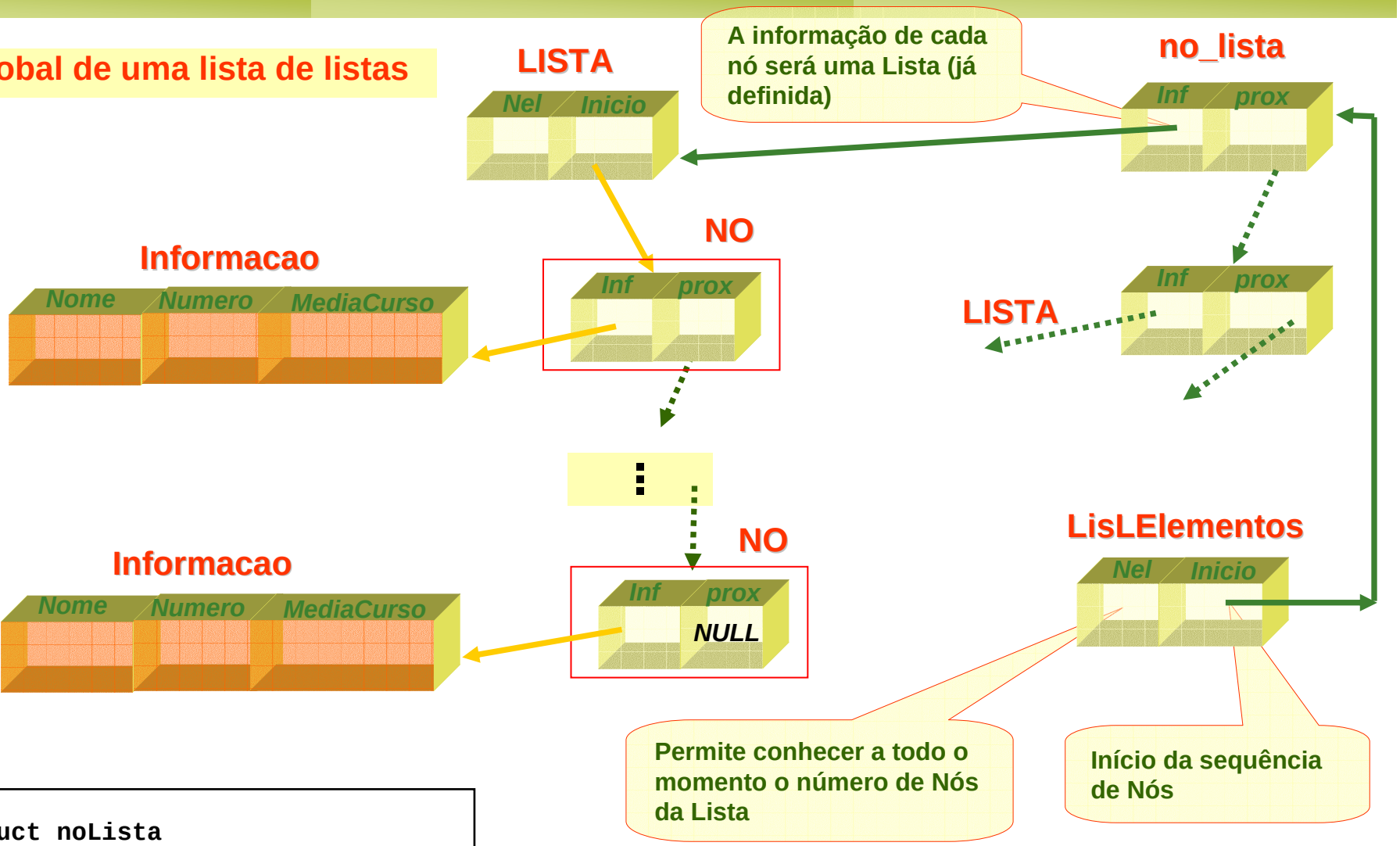
Mostrar todo o conteúdo da lista pela ordem inversa àquela em que se encontram na lista

```
void listarNoContrario(NO p)
{
        if (p)
        {
                listarNoContrario(p->prox);
                mostrarInformacao(p->inf);
        }
}

void listarContrario(LISTA l)
{
        if (l == NULL) return;
        listarNoContrario(l->inicio);
}
```

É utilizada uma função recursiva.
Se cada nó possuísse mais um ponteiro, que apontasse para o nó anterior, tornava-se mais simples (mais **eficiente**?) percorrer a lista nos dois sentidos…



Estrutura global de uma lista de listas




```
typedef struct noLista
{
        LISTA inf;
        struct noLista * prox;
}*NO_LISTA;
```

```
typedef struct
{
        int nElementos;
        NO_LISTA inicio;
}LisLElementos;
```