

Curso de C

Memória Dinâmica

Memória Dinâmica

Objetivos:

- Criar vetores durante a execução do programa e com tamanho exato
- Alterar o tamanho de um vetor
- Criar listas de elementos

Memória Dinâmica

Roteiro:

- Memória dinâmica
- Vetores dinâmicos
- Listas ligadas

Memória Dinâmica

Conceitos

Conceitos: memória **dinâmica** e **estática**

- **Memória estática:**

- Variáveis declaradas no código
- Tamanho fixo
- Número limitado de variáveis

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

- **Memória dinâmica:**

- Espaços adicionais de memória
- Memória alocada/liberada quando necessário

Memória dinâmica:



Conceitos

Passo a passo no programa:

- Solicitar memória dinâmica
- Guardar apontador para espaço obtido
- Armazenar dados
 - Qualquer conteúdo
- Liberar memória obtida

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:

???

Conceitos: obter memória

Chamar função **malloc**

- Parâmetro: tamanho desejado, em bytes
- Resultado: apontador para espaço na memória

```
int *p;  
p = (int*)malloc(4);  
*p = 5;  
printf("%d", *p);
```

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:

5



Conceitos: liberar memória

Chamar função **free**:

- Parâmetro: endereço já alocado
- Resultado: libera a área alocada

```
int *p;  
p = (int*)malloc(4);  
*p = 5;  
free(p);
```

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:

5

Memória Dinâmica

Vetor dinâmico

Vetor dinâmico

Função **sizeof**:

sizeof(variavel)

- Retorna quantidade de memória ocupada pela variável

sizeof(tipo)

- Retorna quantidade de memória ocupada por uma variável com este tipo

```
int a;  
printf ("%d", sizeof(int));  
printf ("%d", sizeof(a));
```

Saída:

4

4

Vetor dinâmico

Passo a passo:

- Calcular tamanho do vetor com **sizeof**
- Reservar memória com **malloc**
- Acessar elementos com **[]** ou **ponteiros**
- Liberar memória do vetor com **free**

Vetor dinâmico

Exemplo:

```
int *v;  
int tamanho;  
  
scanf("%d", &tamanho);  
v = (int*)malloc(sizeof(int)*tamanho);  
  
v[1] = v[2] + v[3];  
  
*(v+1) = *(v+2) + *(v+3);  
  
free(v);
```

Vetor01

Conceito: alterar tamanho de memória

Chamar função **realloc**:

- Parâmetros: apontador para espaço antigo, tamanho desejado
- Resultado: apontador para novo espaço

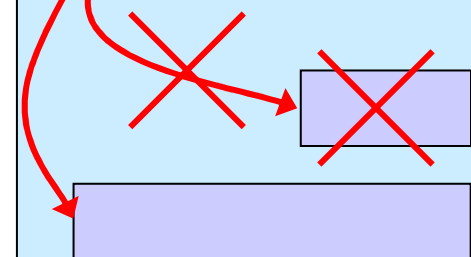
```
int *p;  
p = (int*)malloc(4);  
...  
p = (int*)realloc(p, 8);
```

Programa

Memória estática:

```
int a;  
int b;  
int *p;
```

Memória dinâmica:



Vetor dinâmico

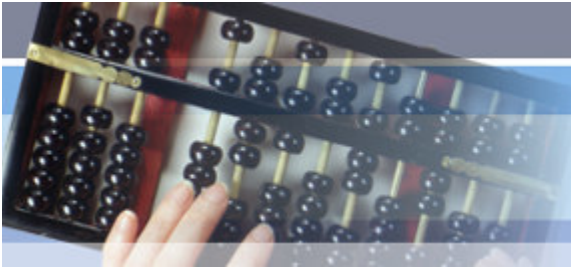
Exemplo:

```
int *v;  
int tamanho, n_tamanho;  
  
scanf("%d", &tamanho);  
v=(int*)malloc(sizeof(int)*tamanho);  
  
scanf("%d", &n_tamanho);  
v=realloc(v, sizeof(int)*n_tamanho);  
  
free(v);
```

Vetor02

Estrutura dinâmica

- Calcular tamanho da estrutura com **sizeof**
- Reservar memória com **malloc**
- Acessar membros da estrutura com **->**
- Liberar memória da estrutura com **free**



Estrutura dinâmica

```
struct complexo {  
    float real;  
    float imaginario;  
}  
  
struct complexo *numero;  
  
numero = (struct complexo *)  
    malloc(sizeof(struct complexo));  
  
numero->real = 10.0;  
numero->imaginario = 5.0;  
  
free(numero);
```

NumComplexo

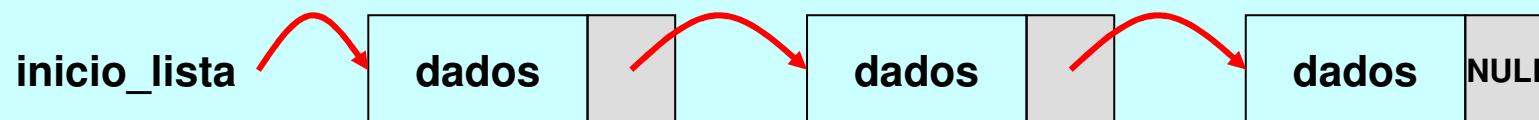
Apontadores

Listas Ligadas

Listas Ligadas

Idéia principal :

- Vetor:
 - Um único grande bloco de memória
 - Elementos consecutivos adjacentes
- Lista ligada:
 - Cada elemento em posição de **memória separada**
 - Posições **desordenadas** e não consecutivas
 - **Apontadores** conectando um elemento ao outro

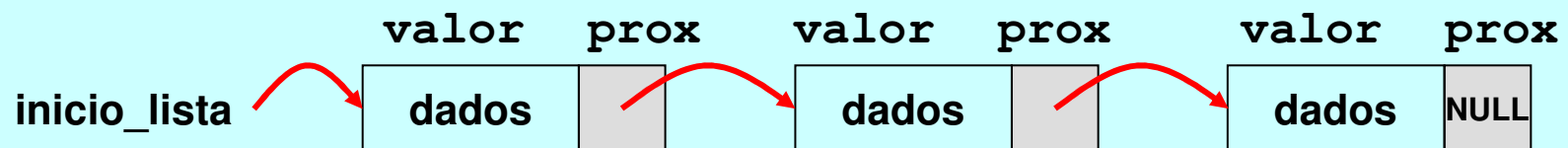


Listas Ligadas: declaração

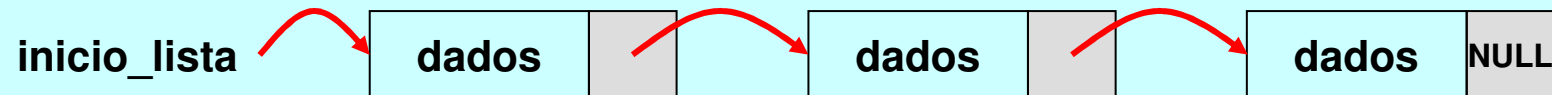
```
struct elemento {  
    int valor;  
    struct elemento *prox;  
}
```

Nome da estrutura

Apontador para próximo elemento na seqüência



Listas Ligadas: percorrer

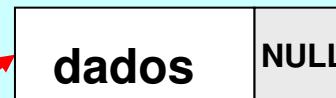


```
struct elemento *atual;  
atual = inicio_lista;  
  
while (atual != null) {  
    printf(" ...", atual->valor);  
  
    atual = atual->proximo;  
}
```

Listas Ligadas: inserir em lista vazia

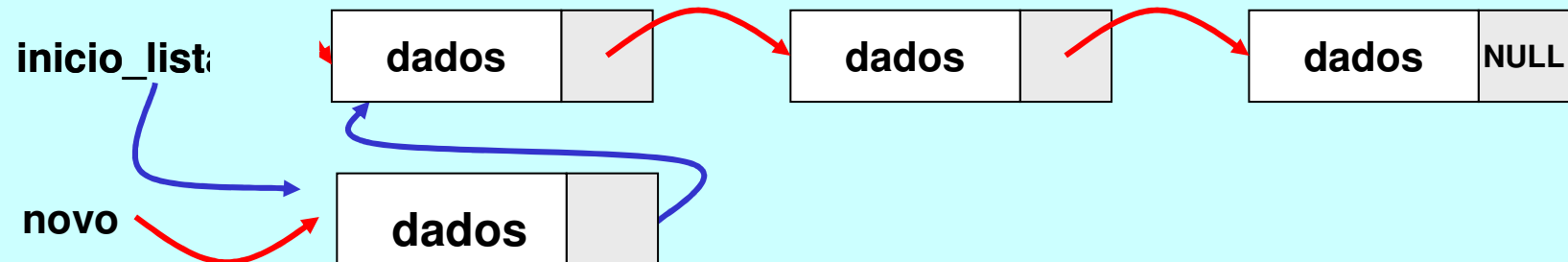
inicio_lista

novo



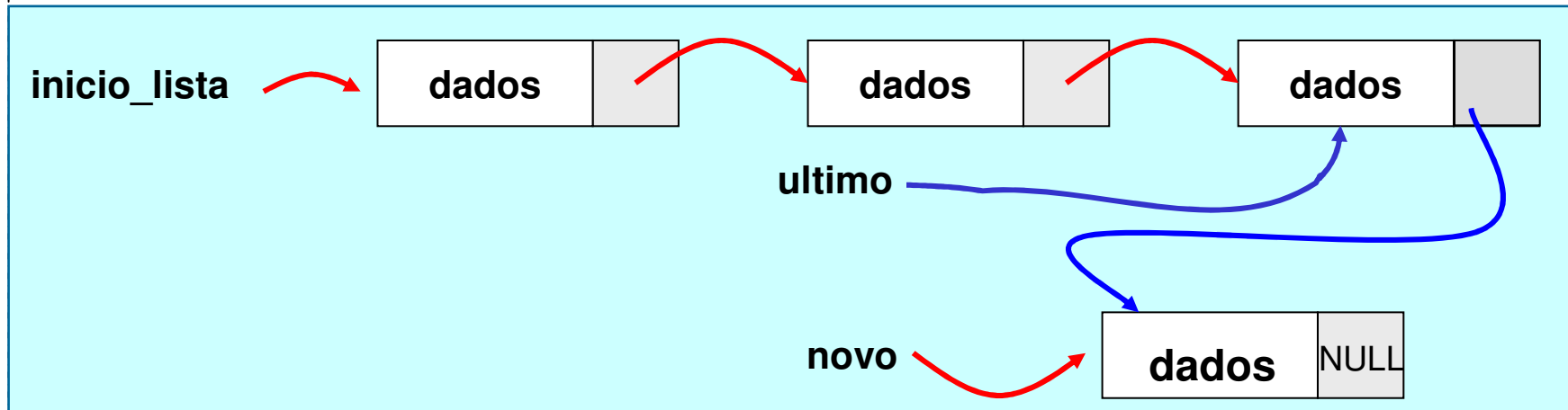
```
novο=(. . . *)malloc(sizeof(. . . ));  
novο->valor = . . . ;  
novο->proximo = NULL;  
inicio_lista = novο;
```

Listas Ligadas: inserir no início, não vazia



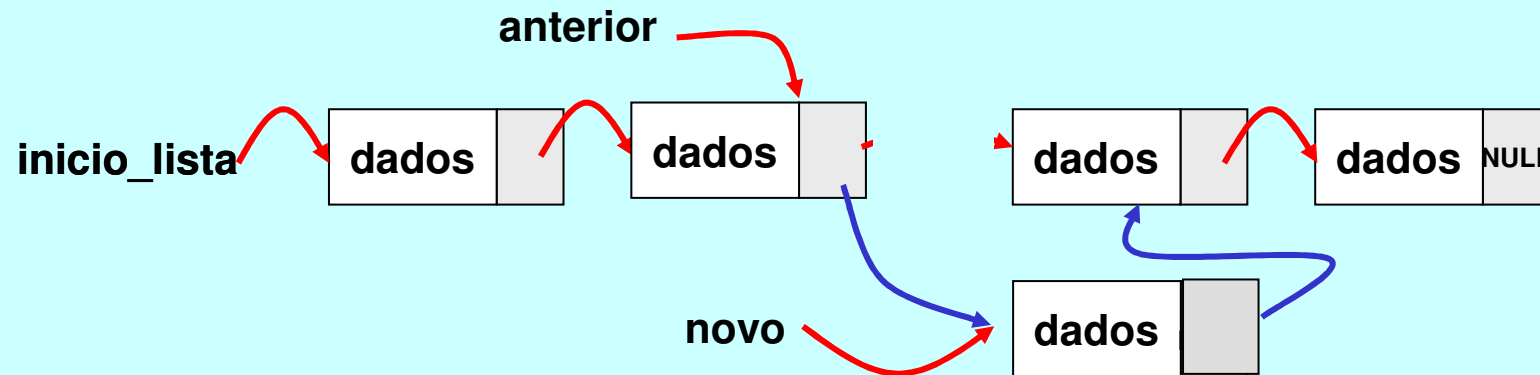
```
novo=(. . . *)malloc(sizeof(. . .));  
novo->valor = . . . ;  
novo->proximo = inicio_lista;  
inicio_lista = novo;
```


Listas Ligadas: inserir no fim



```
struct elemento *ultimo;  
ultimo = inicio_lista;  
while (ultimo->proximo != null) {  
    ultimo = ultimo->proximo; }  
novo=(. . . *)malloc(sizeof(. . .));  
novo->valor = . . . ;  
novo->proximo = NULL;  
ultimo->proximo = novo;
```

Listas Ligadas: inserir no meio (ordenado)

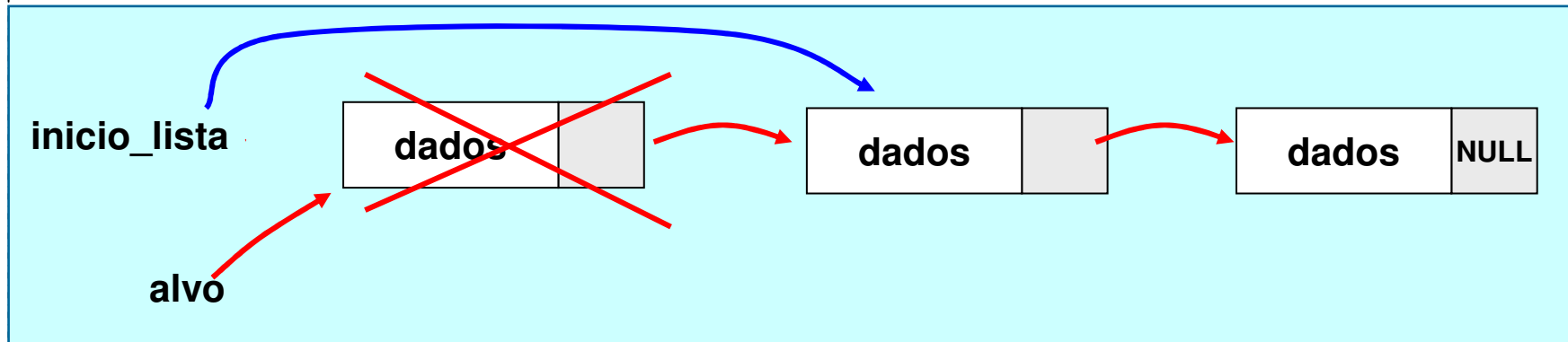


```

novo=(. . . *)malloc(sizeof(. . .));
novo->valor = . . . ;
// ordenada por valor
struct elemento *anterior; anterior = inicio_lista;
while ((anterior-> proximo != NULL)
      && (anterior->proximo->valor < novo->valor)) {
    anterior = anterior->proximo;}
novo->proximo = anterior->proximo;
anterior->proximo = novo;

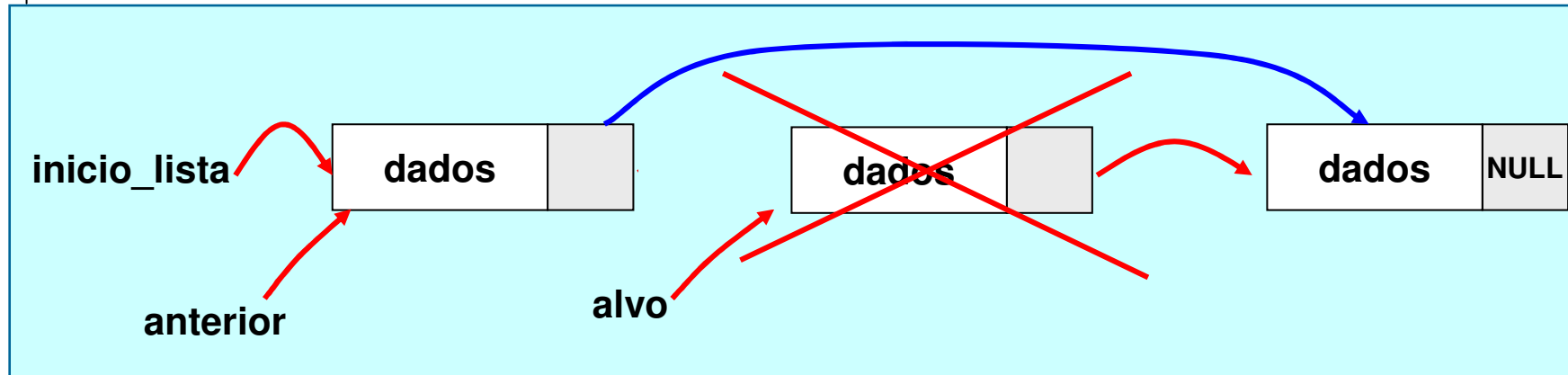
```

Listas Ligadas: remover do inicio (não vazia)



```
alvo = inicio_lista;  
inicio_lista = inicio_lista->proximo;  
// ou: inicio_lista = alvo->proximo;  
free(alvo);
```

Listas Ligadas: remover no meio ou fim

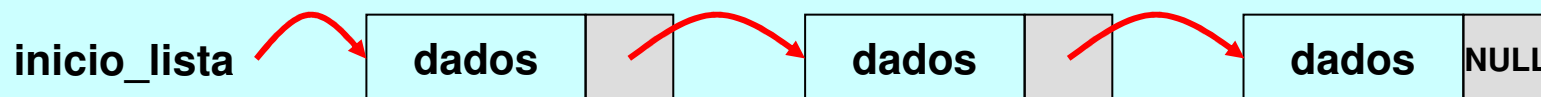


```
// anterior aponta para o elemento  
// imediatamente antes de alvo  
anterior->proximo = alvo->proximo;  
free(alvo);
```

Listas Ligadas

Vantagens/Desvantagens:

- Vantagens:
 - Fácil aumento/diminuição no tamanho da lista
 - Reserva memória exatamente sob medida
 - Permite mais de um critério de ordenação
- Desvantagens:
 - Para cada elemento, armazena um apontador
 - Não é possível acessar diretamente um elemento



Memória Dinâmica

- Vetor2Lista
- Vetor2ListaRec
- ConcatListas
- ConcatListasRec
- InverteLista
- InverteListaRec
- InverteListaNaMemoria
- Vetor2ListaOrd
- Vetor2ListaOrdRec
- OperaListasRec
- OperaComFilas
- Lista2ListaDupla
- Express2Arvore
- DicionarioBinario

Memória Dinâmica

Fim!