Algoritmos e estruturas de Dados

Listas encadeadas

Prof. Maiquel de Brito

Lista

É uma **sequência de elementos**, geralmente do mesmo tipo: L_1, L_2, …, L_N

Uma lista vazia é uma lista com zero elementos

Operações comuns:

- criar uma lista vazia
- adicionar/remover um elemento a uma lista
- determinar a posição de um elemento na lista
- determinar o comprimento (nº de elementos) de uma lista
- concatenar duas listas

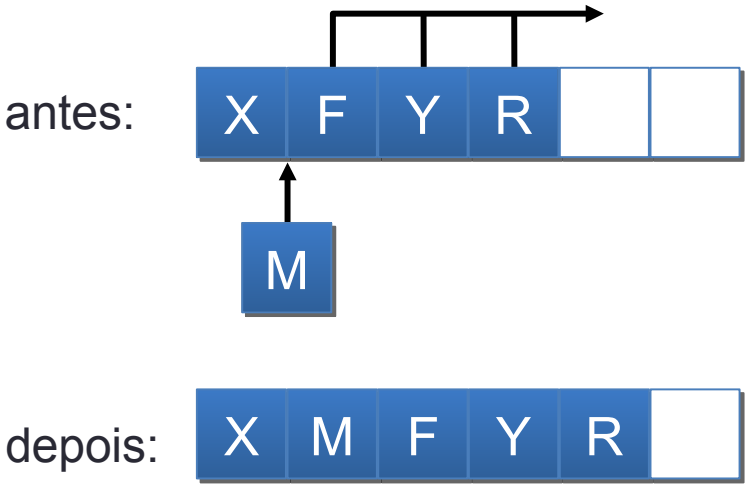
Técnicas de Implementação de Listas

- Baseada em vetores (arrays) dinâmicos
- Baseada em apontadores:
  - Listas ligadas
  - Listas circulares
  - Listas duplamente ligadas

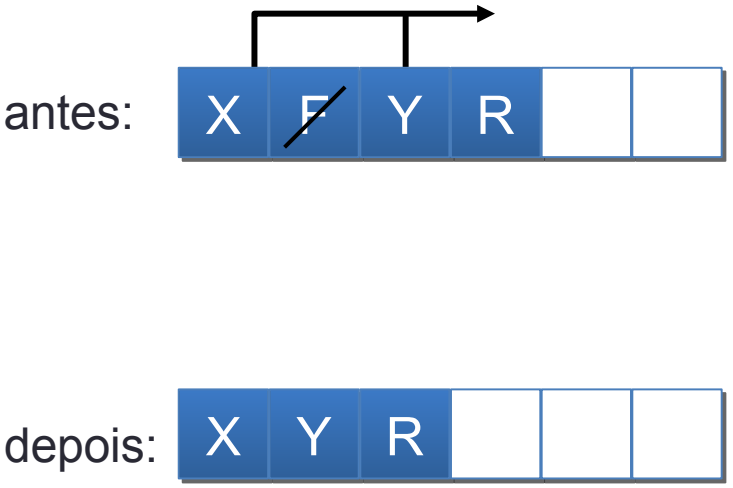
Implementação baseada em Vetores

Os elementos da lista são guardados num **vetor dinâmico.** O vetor é uma estrutura de dados com alocação dinâmica de memória para armazenar N elementos de um dado tipo. O tamanho do vetor exige monitoramento constante. Os itens da lista são armazenados em **posições contíguas** de memória

Inserção de M:



Remoção de F:



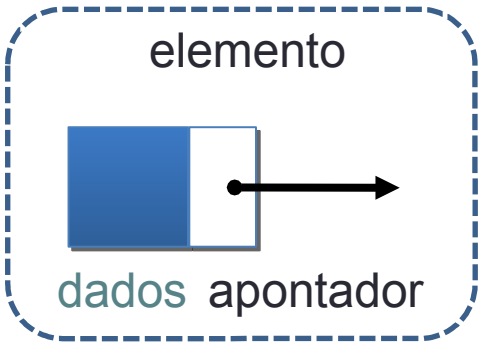
Implementação baseada em Vetores

```c
typedef struct{
  int *elem;
  int tamanho;
}vetor;

void insere_elem(vetor *lista,int val)
{
  lista->tamanho++;
  lista->elem = (realloc(
    lista->elem,lista->tamanho*sizeof(int));
  lista->elem[lista->tamanho-1] = val;

}

vetor cria_lista()
{
    vetor lista;
    lista.elem = malloc(0);
    lista.tamanho = 0;
    return lista;
}
```

```c
int main()
{
    int i;

    printf("Inicio do Programa\n\n");
    vetor lista = cria_lista();

    for(i=10;i>0;i--)
       insere_elem(&lista, i);

    printf("Tamanho vetor:%d\n\n",
                                lista.tamanho);
    for(i=0;i<10;i++)
      printf("Lista.elem[%i] = %d\n", i,
                                lista.elem[i]);
      printf("\n\nFim do Programa\n\n");

}
```

Lista Ligada ou Encadeada

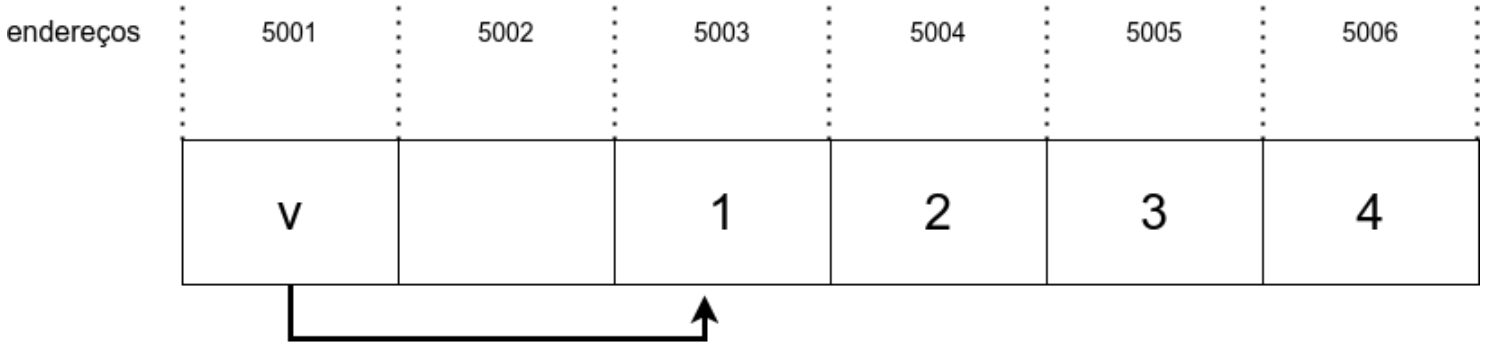
É uma estrutura de dados linear que consiste numa sequência de elementos, em que cada **elemento** inclui um (ou mais) campo(s) com **dados** e um **ponteiro**. O tamanho da lista é facilmente alterado por alocação dinâmica.



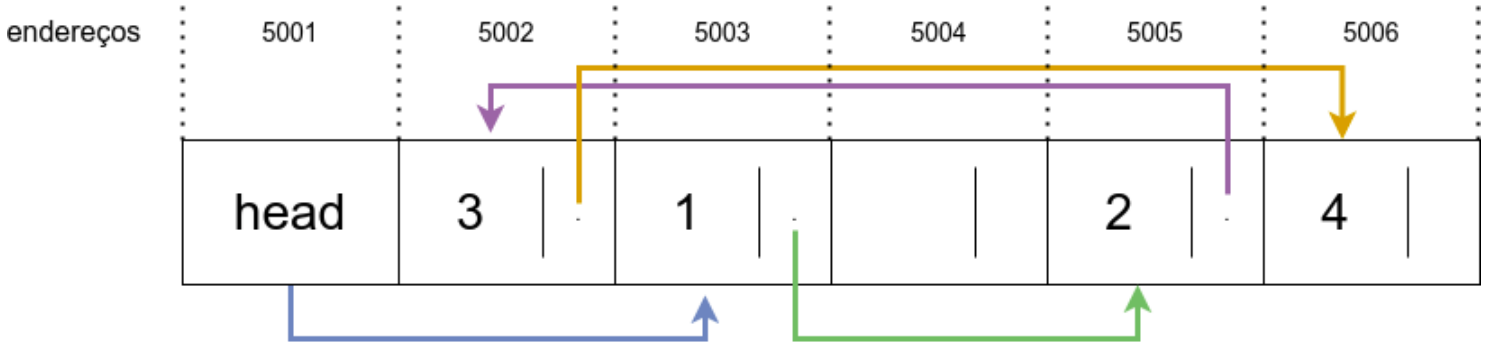
- A alocação de **memória não é contígua**.
- Os **dados** podem ser de **qualquer tipo** e, em geral, os elementos da lista ligada são todos do mesmo tipo de dados.
- Cada elemento inclui um **ponteiro** para o endereço de memória do **próximo elemento** da lista.
- O ponteiro do **último elemento** da lista aponta para **NULL**.
- A lista é acessível através de um apontador "externo", ou **raíz**, que contém o endereço do primeiro elemento da lista.

Alocação Contígua x Dinâmica

Alocação Contígua

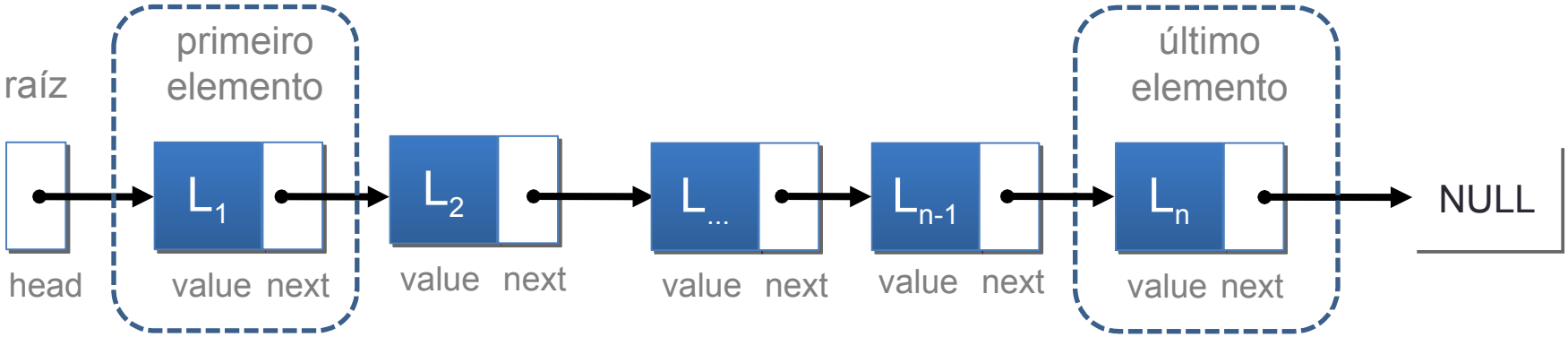


Alocação Dinâmica

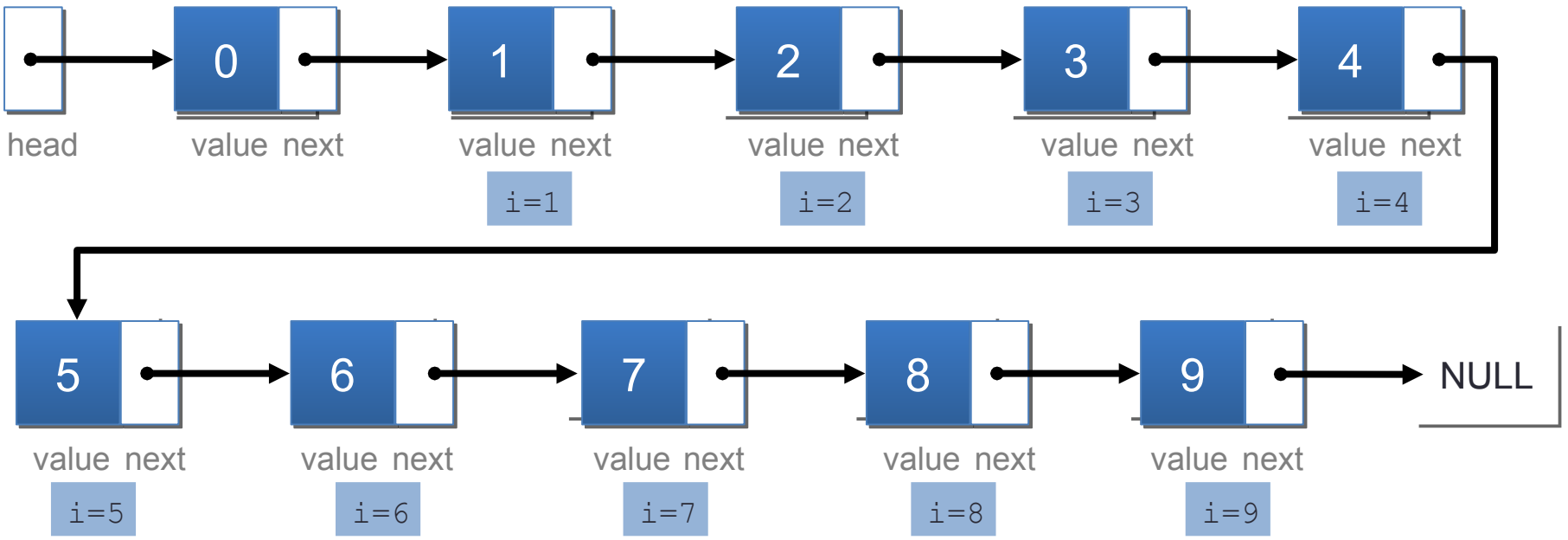


Lista Ligada ou Encadeada

```
typedef struct listItem {
   data_type value;         // valor do elemento da lista
   struct listItem *next; // apontador para o próximo elemento
} ListItem;

ListItem* head = NULL; // apontador para o início da lista (raíz)
```



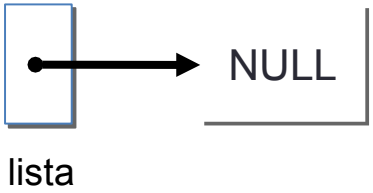
Lista Encadeada (Exemplo)



Lista Encadeada – exemplo : criação de lista

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


//criação de uma lista vazia
ListItem* cria_lista(){
   Return NULL;
}
```

```
int main(){
   ListaItem* lista = cria_lista();
}
```



Uma lista vazia consiste em um ponteiro para um elemento da lista que aponta para *null* (ou seja, não aponta para elemento algum)

Lista Encadeada – exemplo : inserção no início (slide 1)

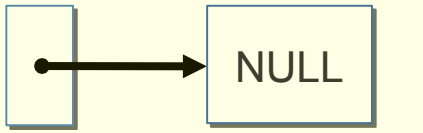
```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);

}
```

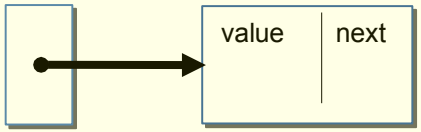

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));




}
```

Aloca memória para armazenar um elemento do tipo `ListItem`

O ponteiro `item` aponta para a porção de memória alocada. Esta porção de memória armazenará o novo elemento.

A porção de memória alocada não está na lista





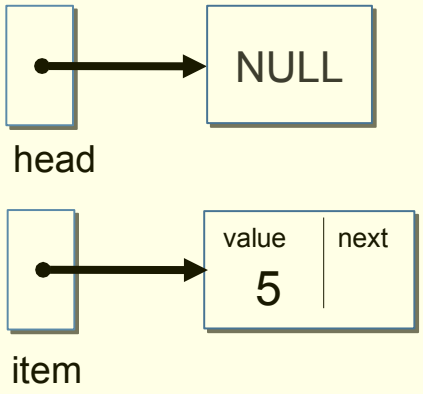
Lista Encadeada – exemplo : inserção no início (slide 2)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);

}
```


```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;




}
```

Atribuição de valores à porção de memória alocada.

A porção de memória alocada ainda não pertence à lista.



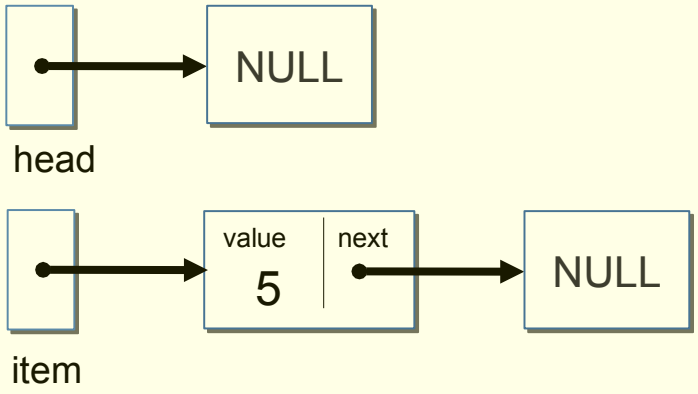
Lista Encadeada – exemplo : inserção no início (slide 3)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);

}
```


```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;



}
```

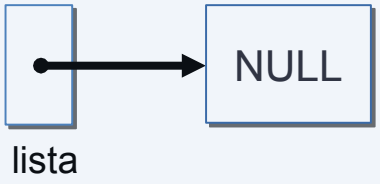
O campo `next` do novo item aponta para a mesma porção de memória que `head` aponta.

A porção de memória alocada ainda não pertence à lista



Lista Encadeada – exemplo : inserção no início (slide 4)

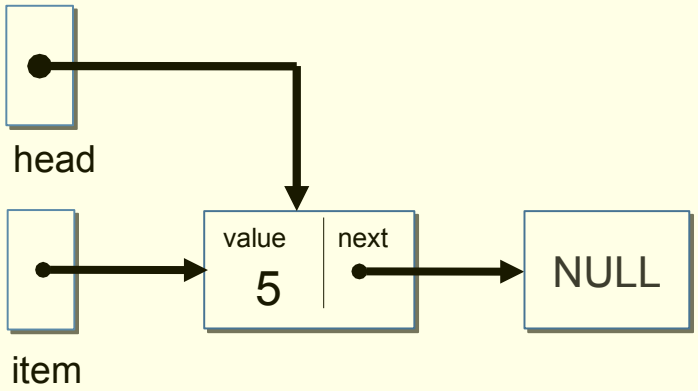
```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);

}
```




```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;



}
```

~~A porção de memória alocada ainda não pertence à lista~~

A cabeça da lista aponta para o novo item, que passa a ser o primeiro elemento da lista..

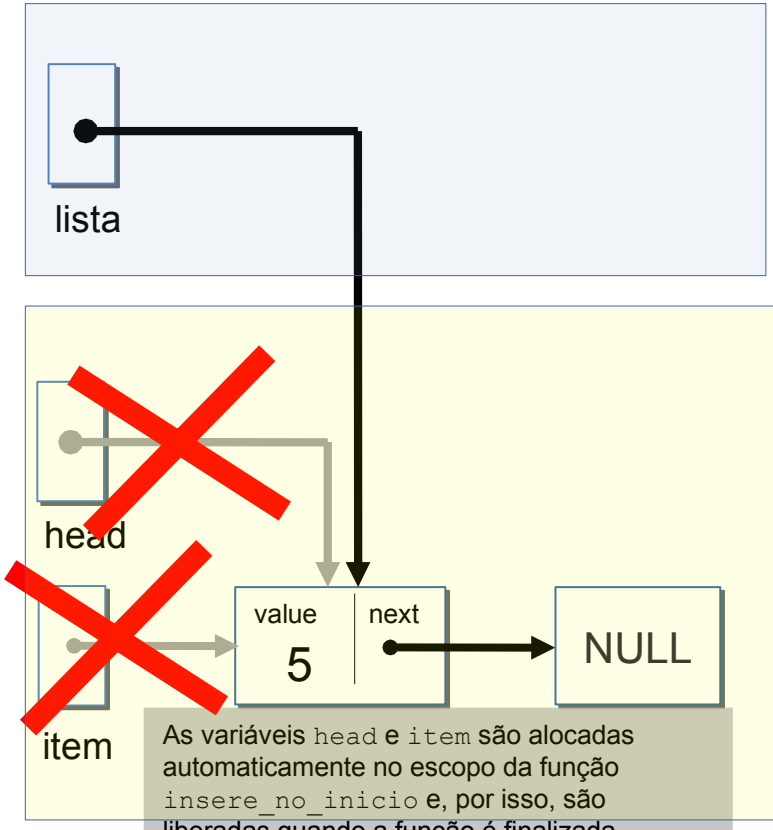


Lista Encadeada – exemplo : inserção no início (slide 5)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);

}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;

   return head;
}
```

A função retorna um ponteiro para o início da lista

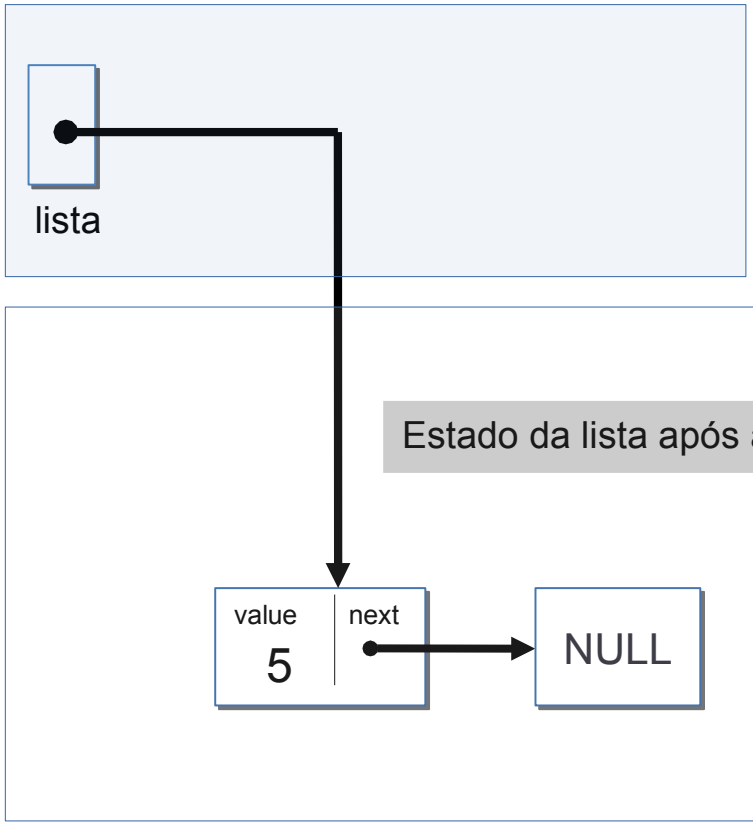


As variáveis `head` e `item` são alocadas automaticamente no escopo da função `insere_no_inicio` e, por isso, são liberadas quando a função é finalizada.

Lista Encadeada – exemplo : inserção no início (slide 6)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);

}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;


   return head;
}
```



Lista Encadeada – exemplo : inserção no início (slide 7)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);
   lista = insere_no_inicio(lista,4);
}
```

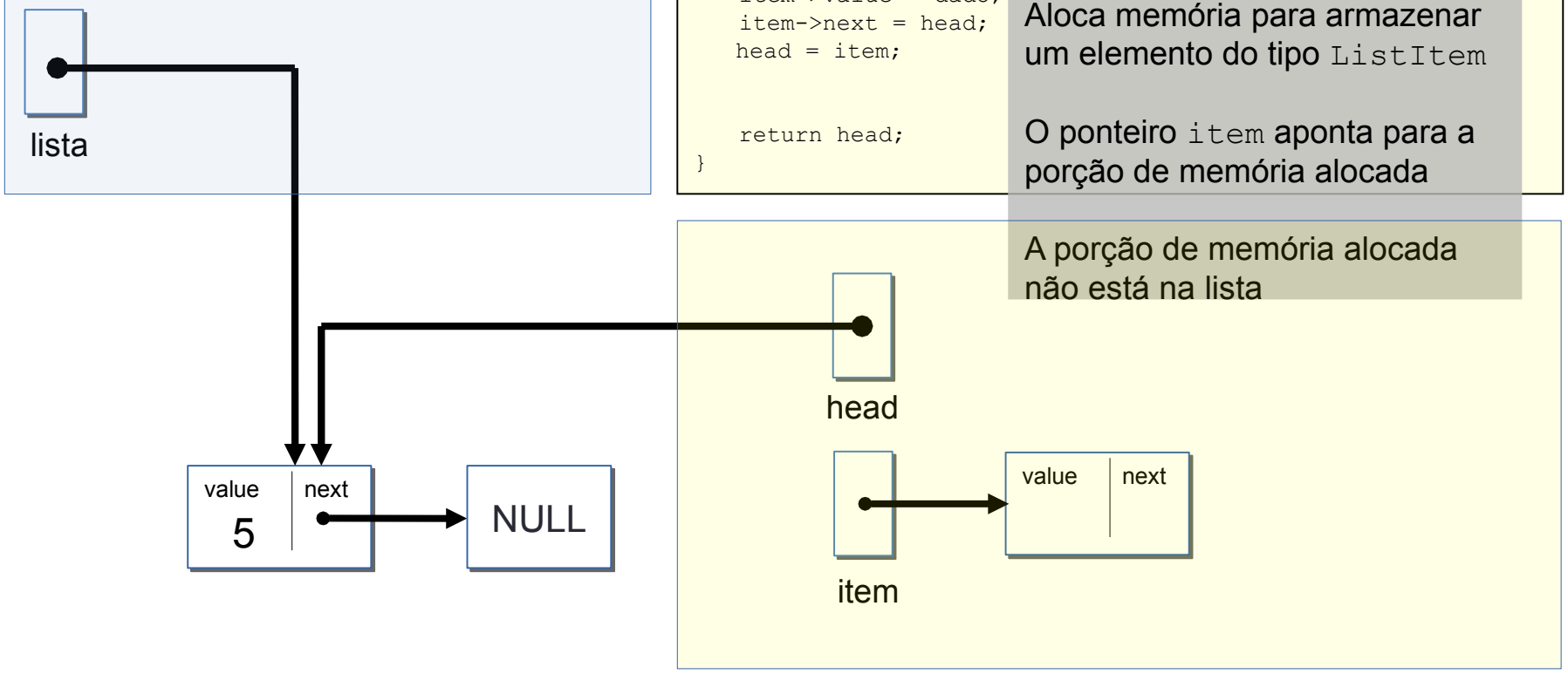
Inserção de um novo elemento na lista

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;


   return head;
}
```

Aloca memória para armazenar um elemento do tipo `ListItem`

O ponteiro `item` aponta para a porção de memória alocada

A porção de memória alocada não está na lista



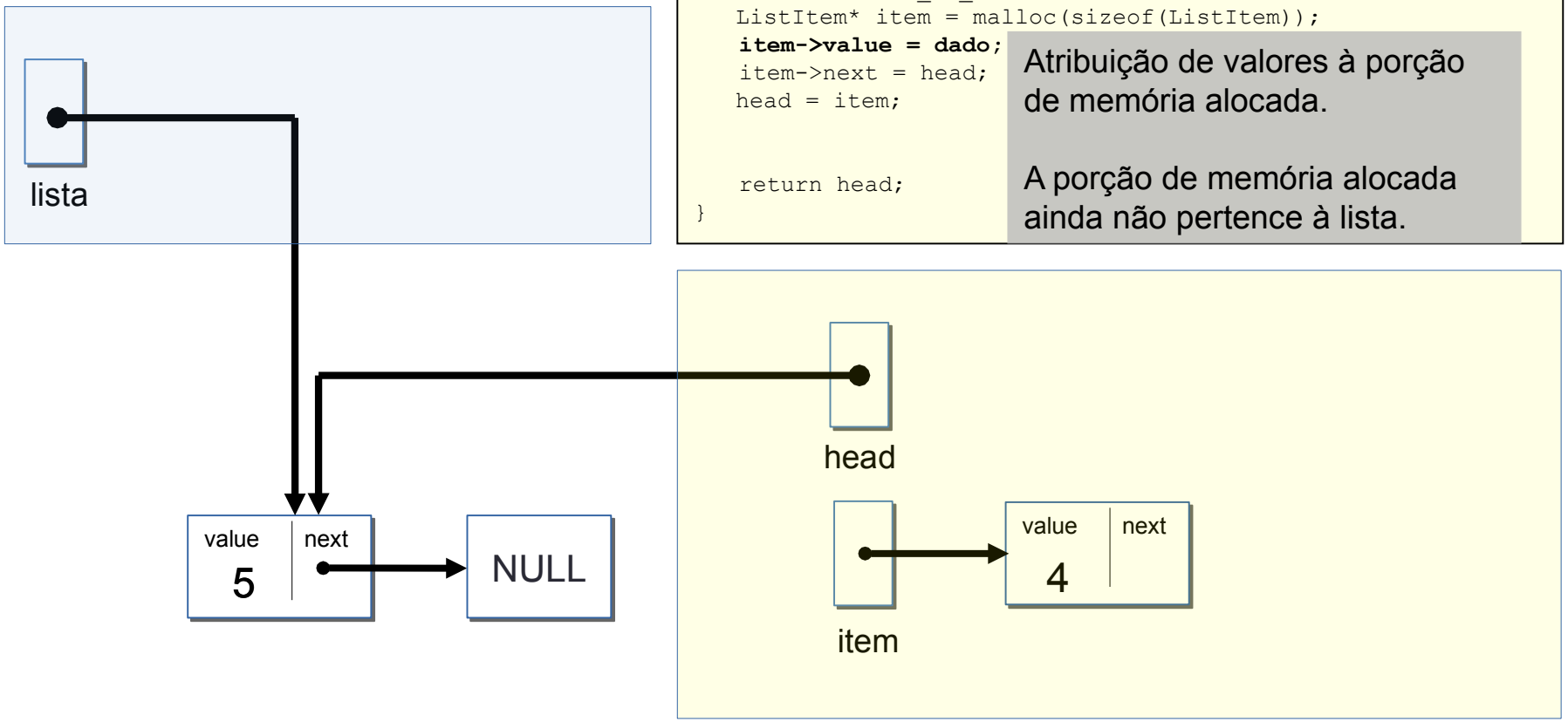
Lista Encadeada – exemplo : inserção no início (slide 8)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);
   lista = insere_no_inicio(lista,4);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;


   return head;
}
```

Atribuição de valores à porção de memória alocada.

A porção de memória alocada ainda não pertence à lista.



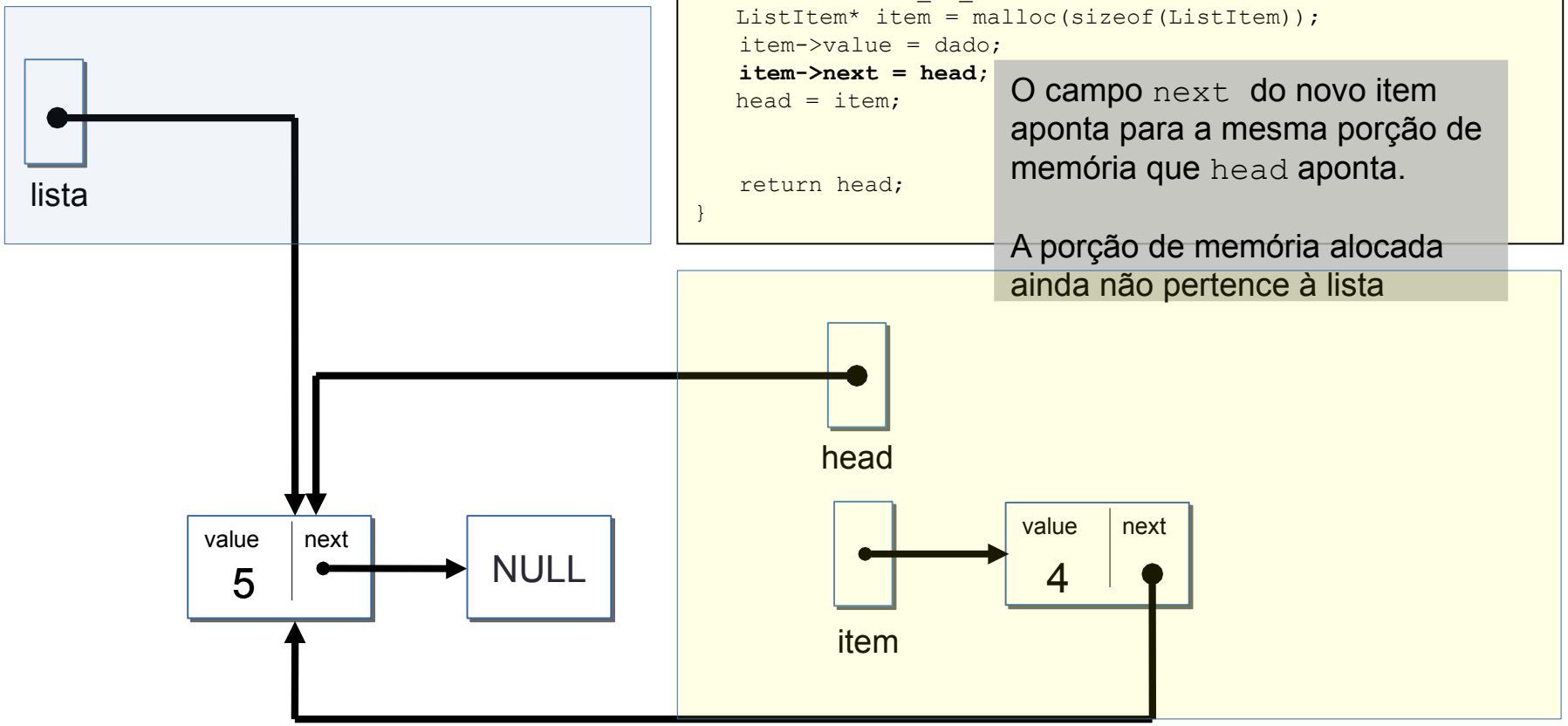
Lista Encadeada – exemplo : inserção no início (slide 9)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);
   lista = insere_no_inicio(lista,4);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;


   return head;
}
```

O campo `next` do novo item aponta para a mesma porção de memória que `head` aponta.

A porção de memória alocada ainda não pertence à lista



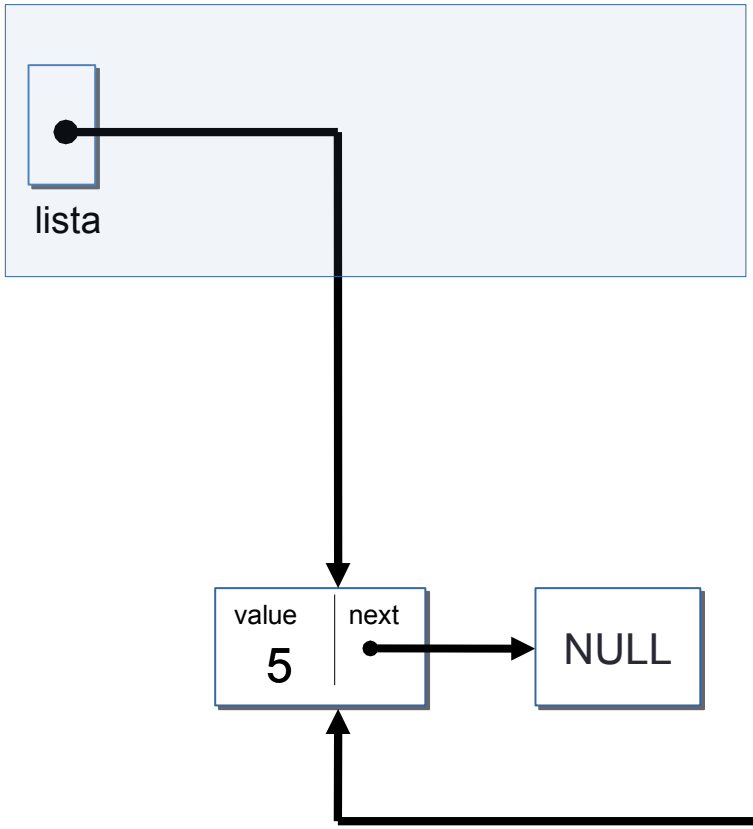
Lista Encadeada – exemplo : inserção no início (slide 10)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);
   lista = insere_no_inicio(lista,4);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;


   return head;
}
```

~~A porção de memória alocada ainda não pertence à lista~~

A cabeça da lista aponta para o novo item, que passa a ser o primeiro elemento da lista..



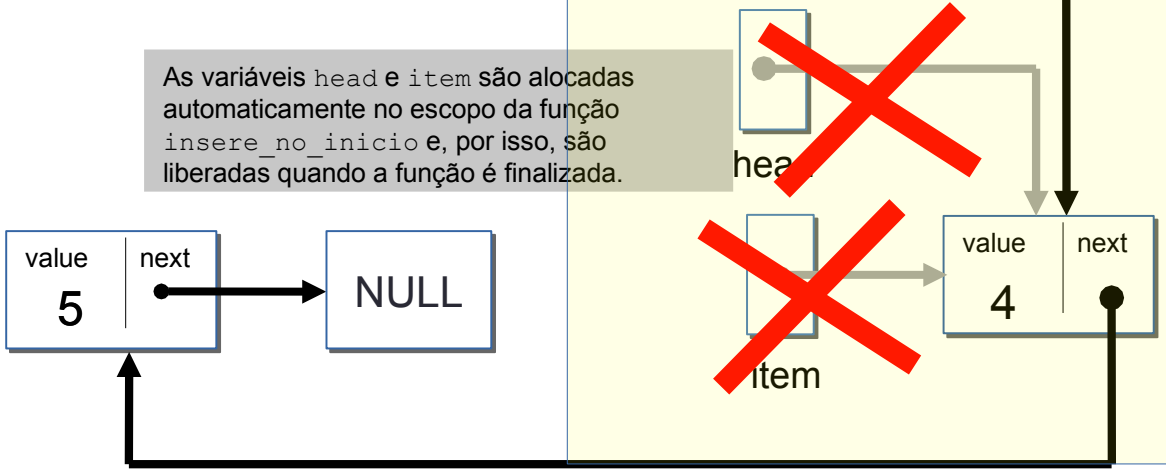
Lista Encadeada – exemplo : inserção no início (slide 11)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);
   lista = insere_no_inicio(lista,4);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;

   return head;
}
```

A função retorna um ponteiro para o início da lista

As variáveis `head` e `item` são alocadas automaticamente no escopo da função `insere_no_inicio` e, por isso, são liberadas quando a função é finalizada.

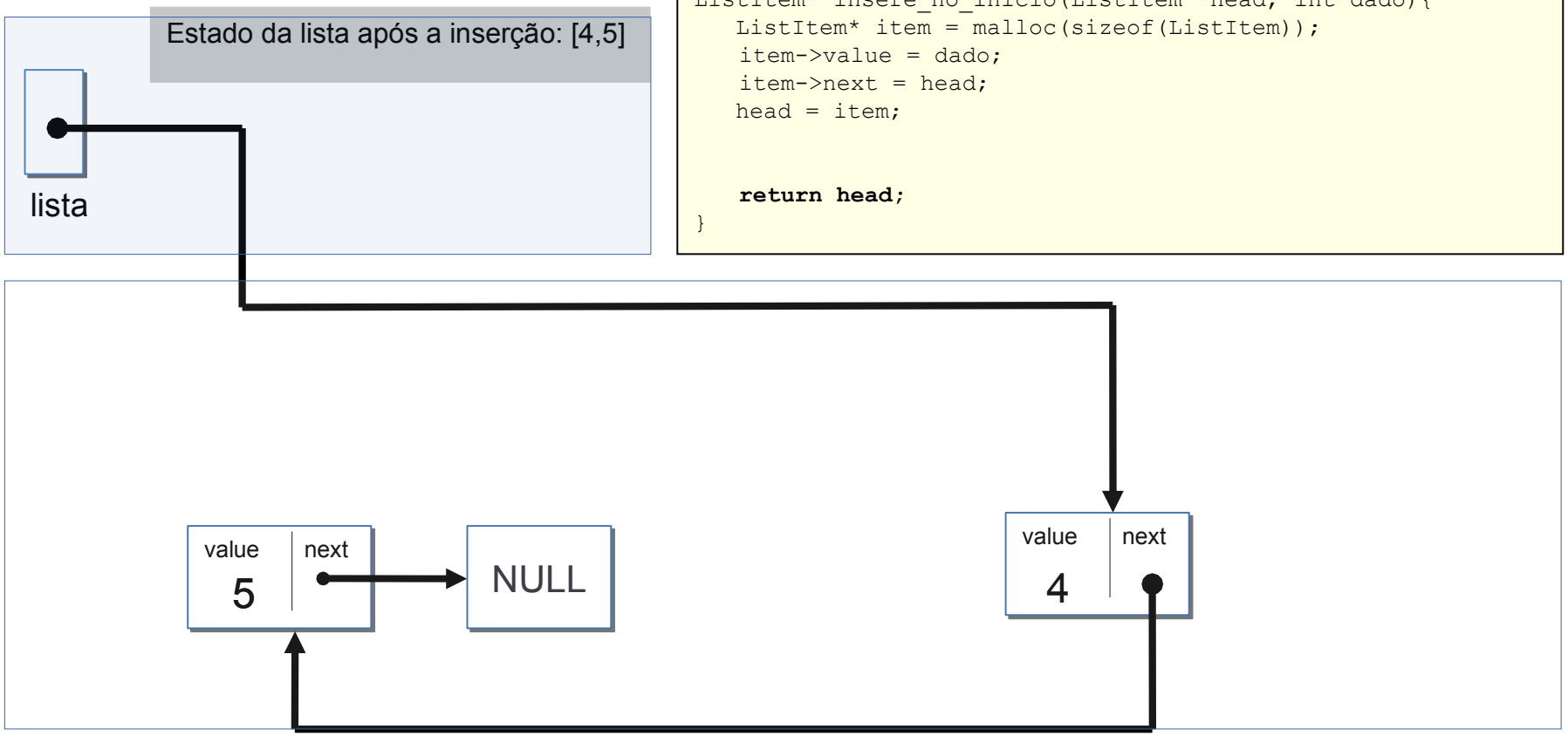


Lista Encadeada – exemplo : inserção no início (slide 12)

```
int main(){
   ListItem* lista = cria_lista()
   lista = insere_no_inicio(lista,5);
   lista = insere_no_inicio(lista,4);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


ListItem* insere_no_inicio(ListItem *head, int dado){
   ListItem* item = malloc(sizeof(ListItem));
   item->value = dado;
   item->next = head;
   head = item;


   return head;
}
```

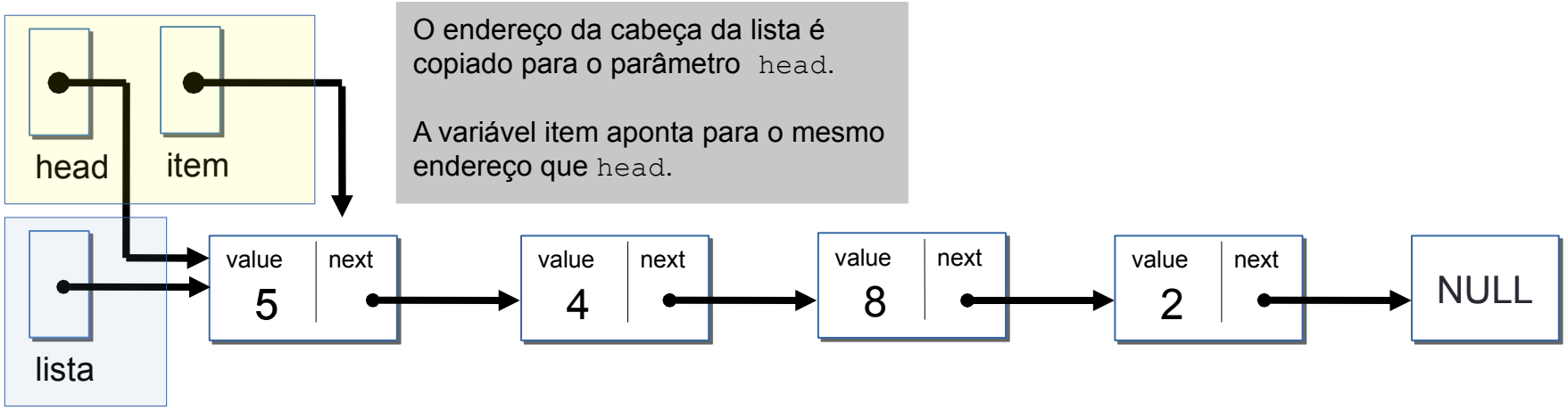
Estado da lista após a inserção: [4,5]



Lista Encadeada – exemplo : iteração sobre a lista (slide 1)

```
int main(){
   ListItem* lista = cria_lista()
   ...
   ...
   print(lista);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


void print(ListItem *head){
   ListItem* item = head;
   while(item!=NULL){
      printf("%d\n",item->value);
      item = item->next;
   }
}
```



O endereço da cabeça da lista é copiado para o parâmetro `head`.

A variável item aponta para o mesmo endereço que `head`.

Lista Encadeada – exemplo : iteração sobre a lista (slide 2)

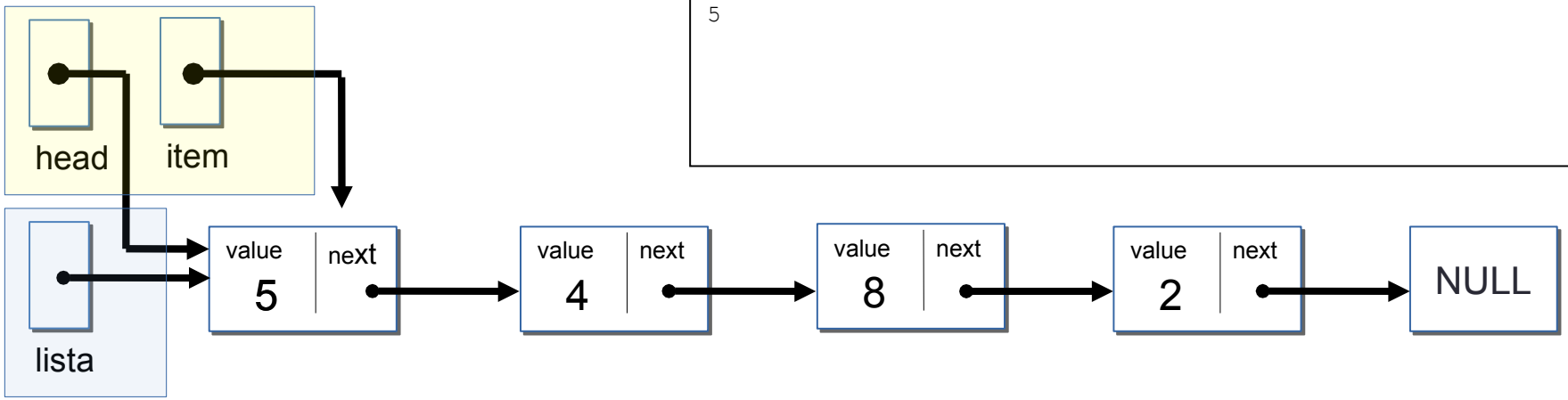
```
int main(){
   ListItem* lista = cria_lista()
   ...
   ...
   print(lista);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


void print(ListItem *head){
   ListItem* item = head;
   while(item!=NULL){
      printf("%d\n",item->value);
      item = item->next;
   }
}
```

Imprime o campo `value` armazenado na porção de memória para onde `item` aponta.

Saída:

```
5
```



Lista Encadeada – exemplo : iteração sobre a lista (slide 2)

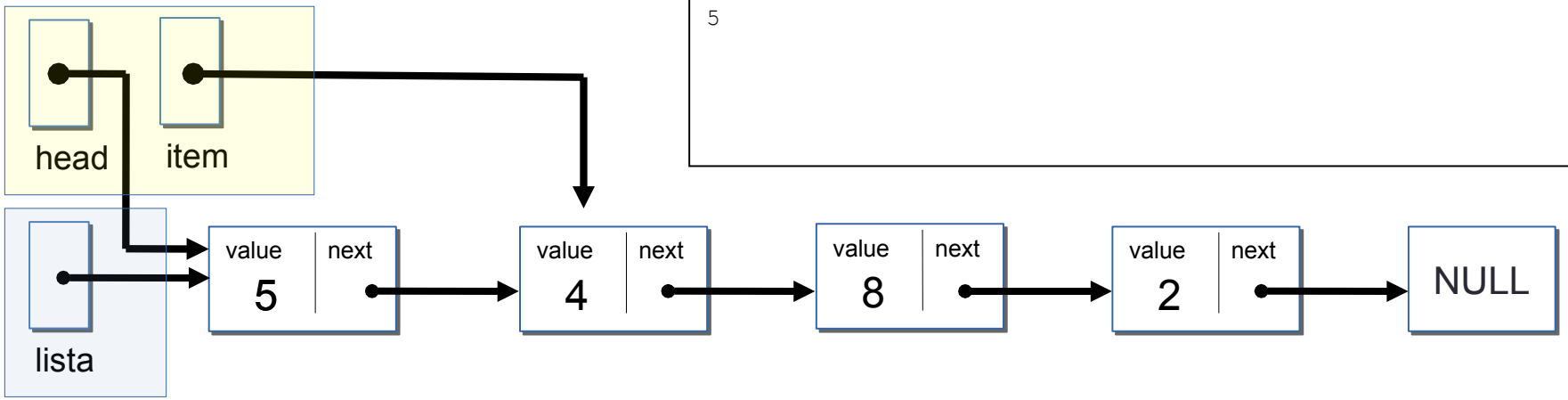
```
int main(){
   ListItem* lista = cria_lista()
   ...
   ...
   print(lista);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


void print(ListItem *head){

   while(item!=NULL){
      printf("%d\n",item->value);
      item = item->next;
   }
}
```

O ponteiro `item` passa a apontar para o próximo elemento.

Saída:

```
5
```



Lista Encadeada – exemplo : iteração sobre a lista (slide 2)

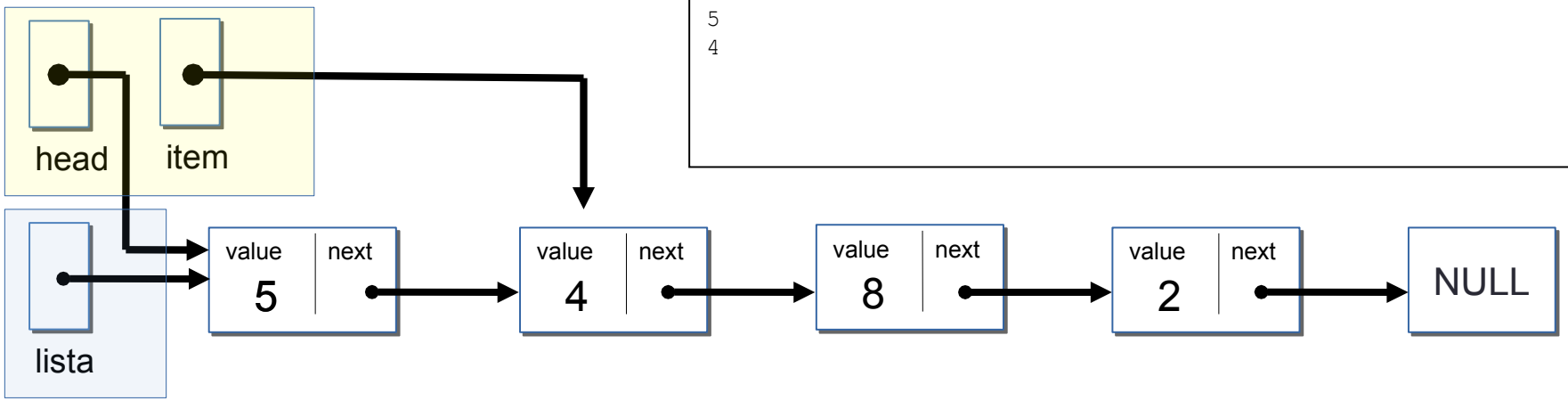
```
int main(){
   ListItem* lista = cria_lista()
   ...
   ...
   print(lista);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


void print(ListItem *head){

   while(item!=NULL){
      printf("%d\n",item->value);
      item = item->next;
   }
}
```

Imprime o campo `value` armazenado na porção de memória para onde `item` aponta.

Saída:

```
5
4
```



Lista Encadeada – exemplo : iteração sobre a lista (slide 2)

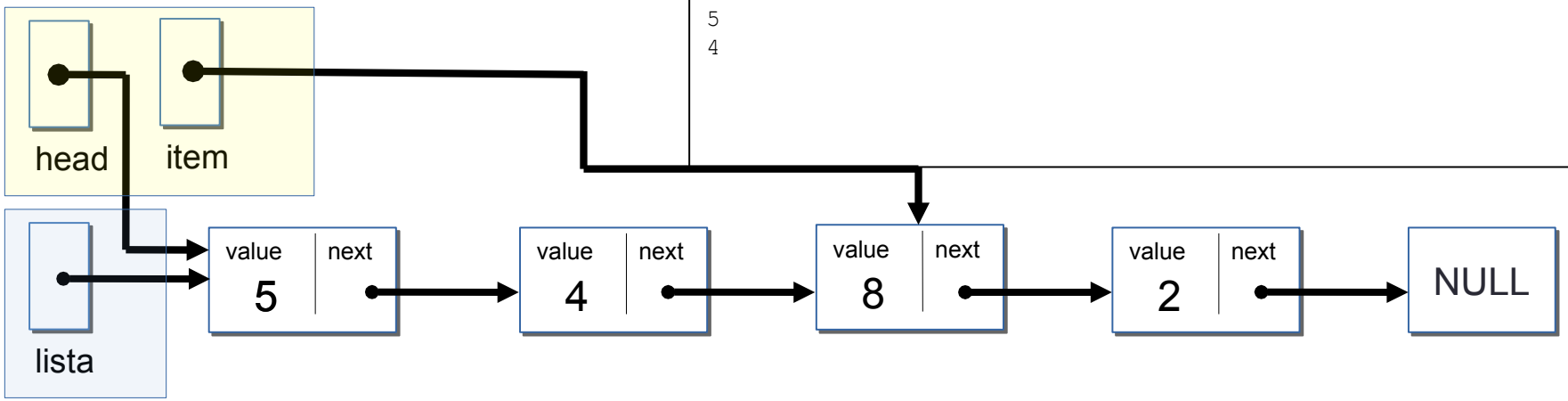
```
int main(){
   ListItem* lista = cria_lista()
   ...
   ...
   print(lista);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


void print(ListItem *head){

   while(item!=NULL){
      printf("%d\n",item->value);
      item = item->next;
   }
}
```

O ponteiro `item` passa a apontar para o próximo elemento.

Saída:

```
5
4
```



Lista Encadeada – exemplo : iteração sobre a lista (slide 2)

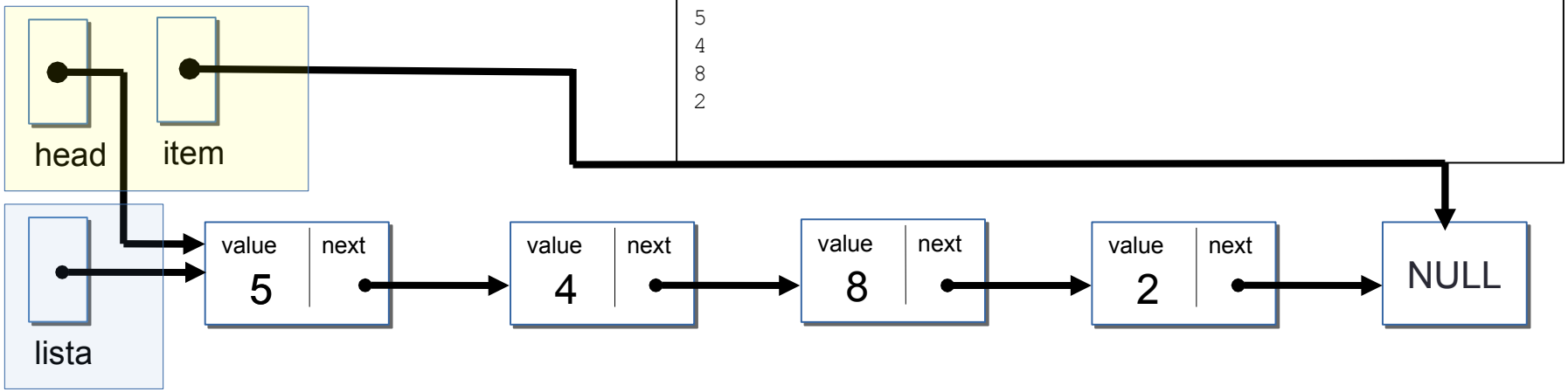
```
int main(){
   ListItem* lista = cria_lista()
   ...
   ...
   print(lista);
}
```

```
typedef struct listItem {
   int value; // valor do elemento da lista
   struct listItem *next; // ponteiro para o próximo elemento
} ListItem;


void print(ListItem *head){

   while(item!=NULL){
      printf("%d\n",item->value);
      item = item->next;
   }
}
```

Continua a iteração até `item` apontar para `NULL`

Saída:

```
5
4
8
2
```



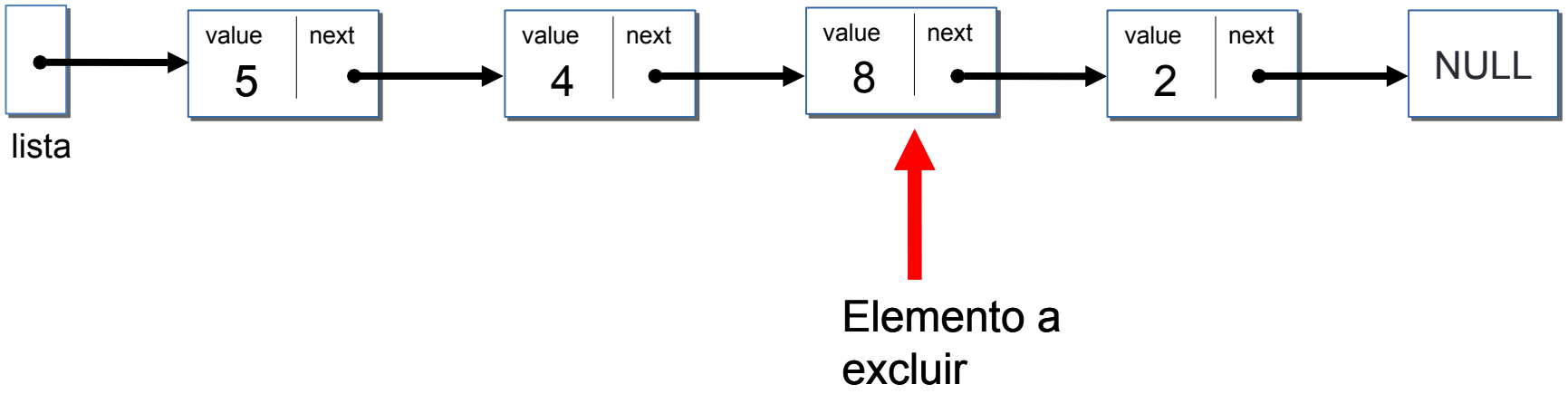
Lista Encadeada – exemplo : remoção de elementos da lista (slide 1)

A remoção de elementos em listas encadeadas segue o mesmo princípio da inserção:

Deve-se ajustar os ponteiros dos elementos para que o elemento a ser removido não faça mais parte da lista.

Exemplo – exclusão do 3º elemento da lista:

Passo 1: encontrar o elemento a ser excluído (usando a estratégia de iteração sobre a lista)



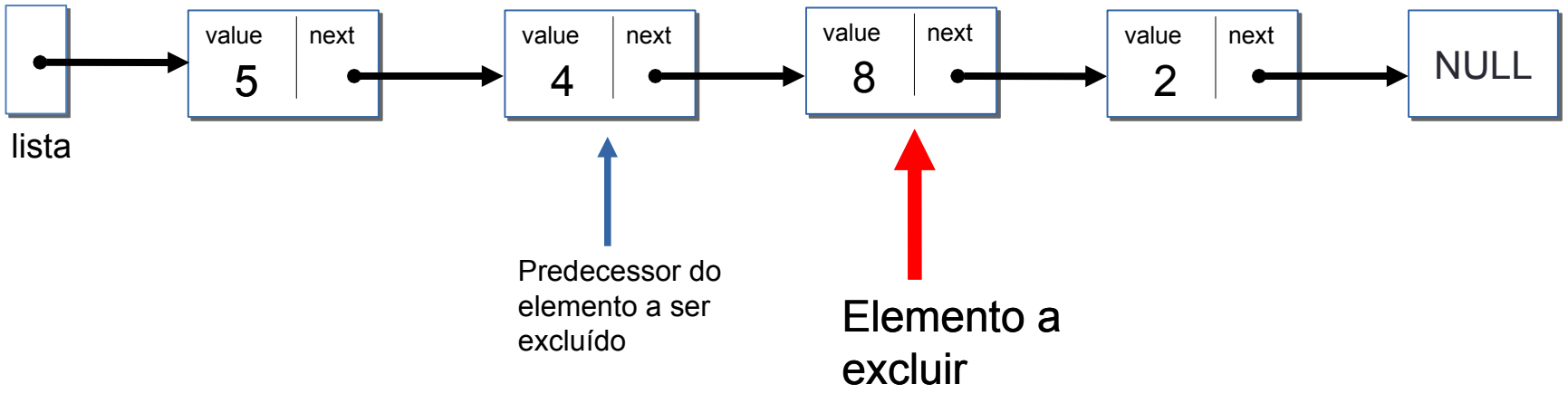
Lista Encadeada – exemplo : remoção de elementos da lista (slide 1)

A remoção de elementos em listas encadeadas segue o mesmo princípio da inserção:

Deve-se ajustar os ponteiros dos elementos para que o elemento a ser removido não faça mais parte da lista.

Exemplo – exclusão do 3º elemento da lista:

Passo 1: encontrar o elemento a ser excluído (usando a estratégia de iteração sobre a lista)
Passo 2: encontrar o predecessor do elemento a ser excluído



Lista Encadeada – exemplo : remoção de elementos da lista (slide 1)

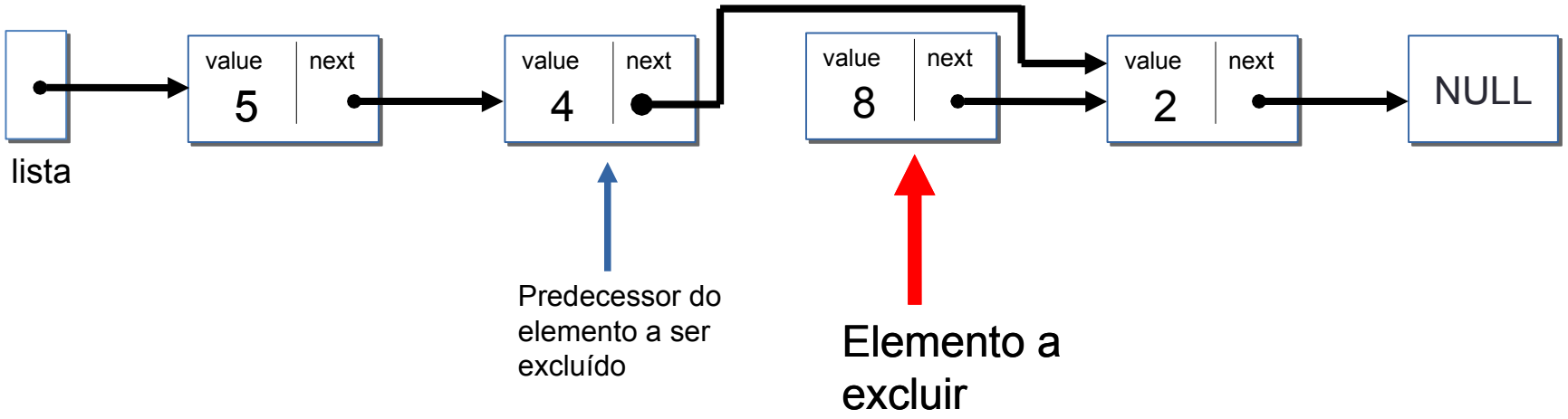
A remoção de elementos em listas encadeadas segue o mesmo princípio da inserção:

Deve-se ajustar os ponteiros dos elementos para que o elemento a ser removido não faça mais parte da lista.

Exemplo – exclusão do 3º elemento da lista:

Passo 1: encontrar o elemento a ser excluído (usando a estratégia de iteração sobre a lista)
Passo 2: encontrar o predecessor do elemento a ser excluído
Passo 3: fazer o predecessor apontar para o sucessor do elemento a ser excluído

observações: - o elemento a ser excluído já não faz mais parte da lista
o elemento s ser excluído continua apontando para o seu sucessor



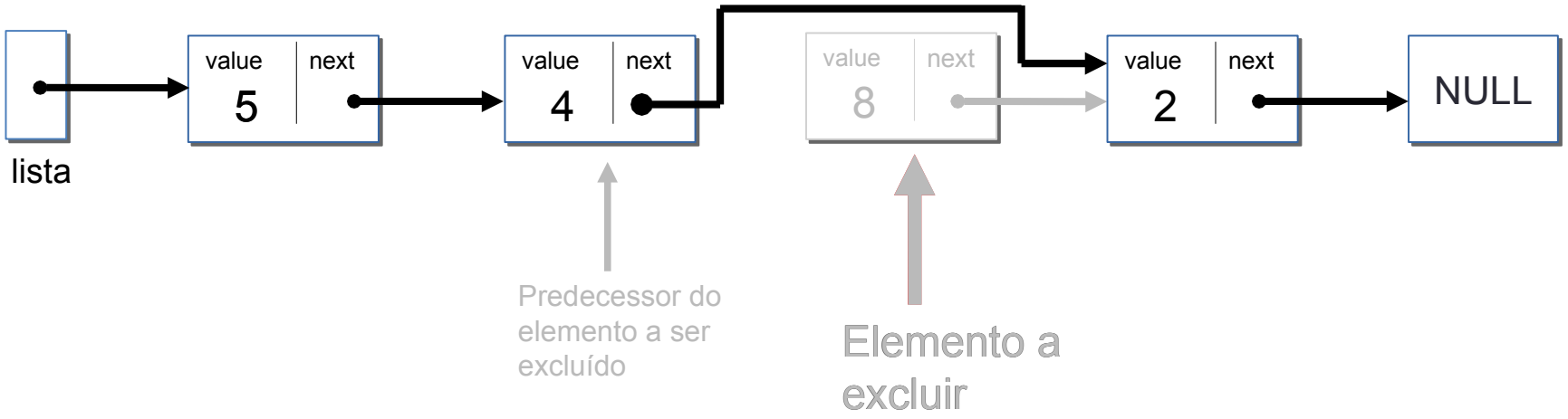
Lista Encadeada – exemplo : remoção de elementos da lista (slide 1)

A remoção de elementos em listas encadeadas segue o mesmo princípio da inserção:

Deve-se ajustar os ponteiros dos elementos para que o elemento a ser removido não faça mais parte da lista.

Exemplo – exclusão do 3º elemento da lista:

Passo 1: encontrar o elemento a ser excluído (usando a estratégia de iteração sobre a lista)
Passo 2: encontrar o predecessor do elemento a ser excluído
Passo 3: fazer o predecessor apontar para o sucessor do elemento a ser excluído
Passo 4: liberar a memória do elemento excluído (em linguagem C, usar a função *free*)



Vector Dinâmico vs. Lista Ligada

Implementação baseada em Vetores (desvantagens)

- Para inserir ou eliminar um elemento no vetor poderá ser necessário:
  - mover outros elementos para posições posteriores ou anteriores (no pior caso, todos os elementos!)
  - re-alocar memória, quando a capacidade atual do vector é excedida; ou diminuir o tamanho do vector, quando se eliminam muitos elementos, evitando ocupação desnecessária de memória

Vector Dinâmico vs. Lista Ligada

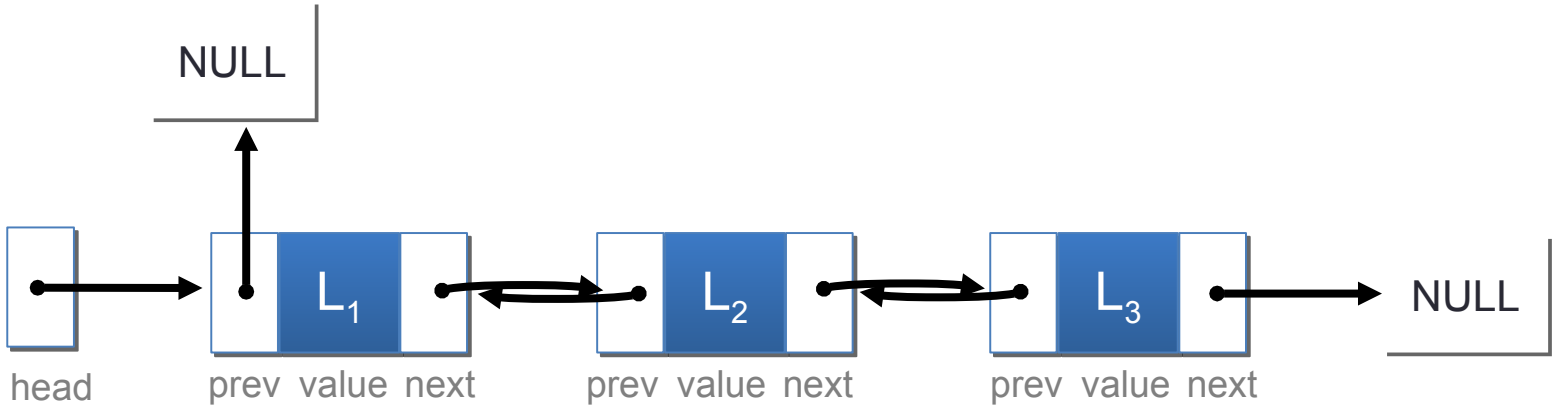
Implementação baseada em Listas Ligadas (desvantagens)

- Não é possível aceder a posições aleatórias da lista através do uso de índices.
- Os elementos requerem mais espaço de memória do que na implementação baseada em vetores porque, para além dos dados, é necessário armazenar a referência do próximo elemento.

Lista Duplamente Ligada

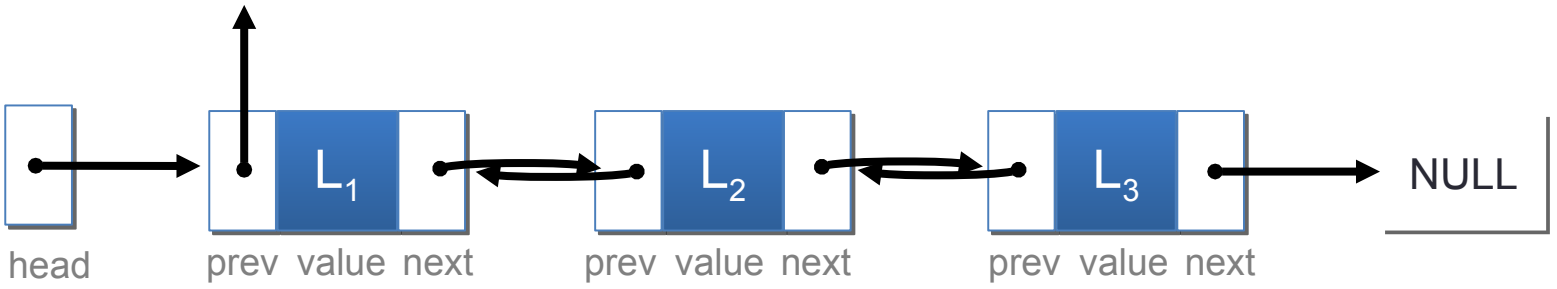
Uma **lista duplamente ligada** é um tipo especial de lista, em que cada elemento inclui dois apontadores, um para o elemento anterior e outro para o elemento seguinte da lista.

Estas listas são úteis em aplicações em que há necessidade de percorrer a lista em ambos os sentidos.



Lista Duplamente Ligada

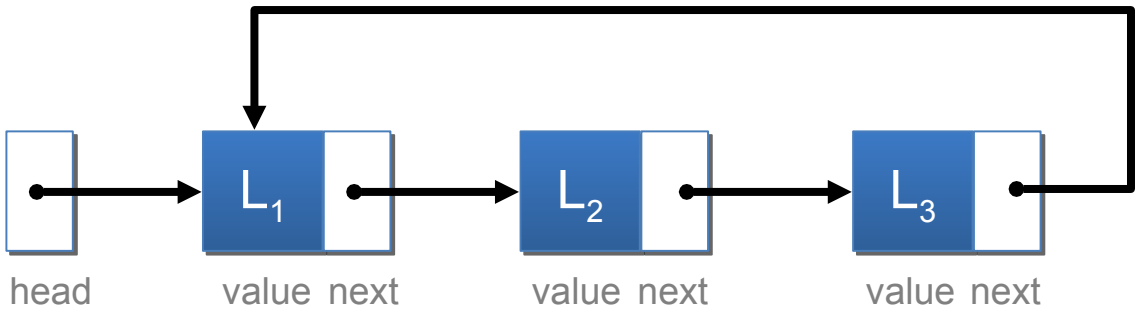
```
struct listItem {
   data_type value;         /* valor do elemento da lista */
   struct listItem *prev; /* apontador para o elemento anterior */
   struct listItem *next; /* apontador para o próximo elemento */
};

struct listItem* head = NULL; /*apontador para o início da lista (raiz) */
                              /* pode também definir-se um apontador
                                 para o fim da lista */
```



Listas Circulares

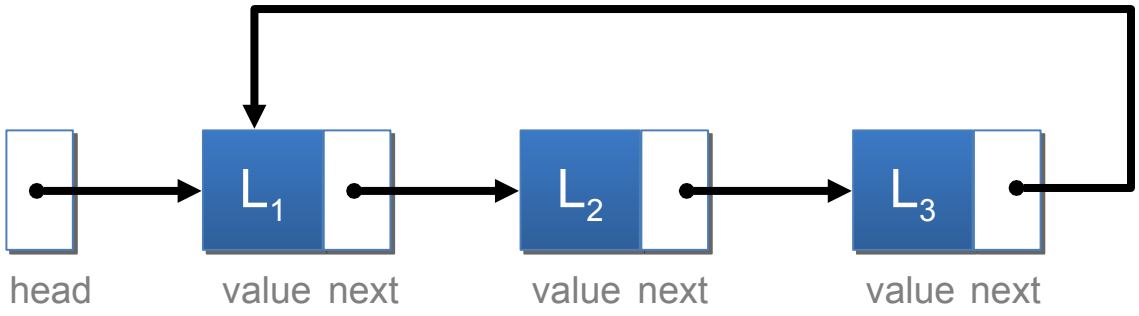
Uma **lista circular** é um tipo especial de lista ligada, em que o último elemento referencia o primeiro elemento da lista.

Estas listas são úteis em aplicações em que há necessidade de percorrer a lista em modo “loop”, dado que permitem percorrer a lista do fim para o início.



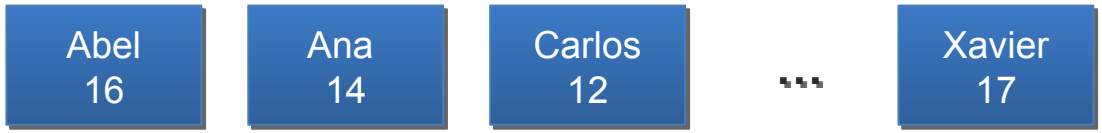
Listas Circulares

```
struct listItem {
   data_type value;         /* valor do elemento da lista */
   struct listItem *next; /* apontador para o próximo elemento */
};

struct listItem* head = NULL; /* apontador para o início da lista (raíz) */
                              /* pode também definir-se um apontador para o
                                 fim da lista */
```



Exemplo de uma Lista

- Lista dos alunos inscritos a Algoritmos e estruturas de Dados



- A lista guarda informação sobre o nome e nota dos alunos

- Operações:
  - adicionar um aluno à lista
  - remover alunos da lista
  - listar todos os alunos e respectivas notas
  - pesquisar um aluno pelo nome
  - ordenar a lista por ordem alfabética (ou nota)
  - etc.

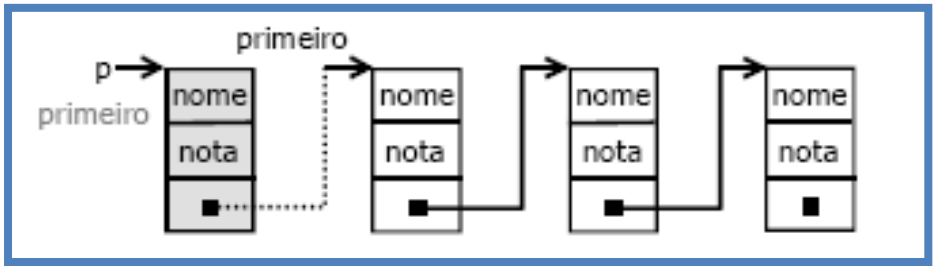
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NCAR 100

typedef struct _aluno {
  char nome[NCAR];
  int nota;
  struct _aluno *proximo;
} aluno;
```

```c
typedef struct {
  aluno *primeiro;
} lista;


lista* init(){
  lista *l = malloc(sizeof(lista));
  l->primeiro = NULL;
  return l;
}
```

```c
void ler_str(char *s, int n) { /* ler um string */
  fgets(s, n, stdin);
  s[strlen(s)-1] = '\0';
}

void ler_aluno(aluno *a) { /* ler um registo de aluno */
   printf("Nome: "); ler_str(a->nome, NCAR);
   printf("Nota: "); scanf("%d", &a->nota);
}

void escrever_aluno(aluno *a) { /* escrever um registo de aluno */
  printf("Nome: %s\n", a->nome);
  printf("Nota: %2d\n", a->nota);
}
```

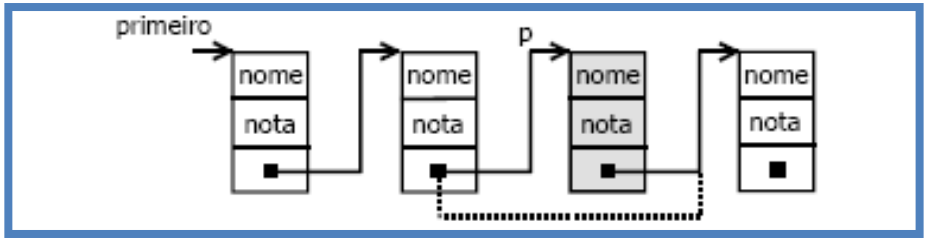
```c
aluno* novo_aluno(void) {
  aluno *p;
  p = malloc(sizeof(aluno));
  p->proximo = NULL;
  return p;
}

void insere(lista * l) { /* insere um novo registo no inicio da lista */
  aluno *p;
  char s[NCAR];

  p = novo_aluno();
  ler_str(s, NCAR);
  ler_aluno(p);

  if(l->primeiro == NULL)
    l->primeiro = p;
  else
  {
    p->proximo = l->primeiro;
    l->primeiro = p;
  }
}
```

Inserção no início da lista



```
aluno* busca(lista * l) { /* efetua a busca de um nome na lista */
  aluno *p;
  char s[NCAR];
  printf("Nome a procurar? "); ler_str(s, NCAR); ler_str(s, NCAR);
  p = l->primeiro;
  while(p != NULL && strcmp(p->nome, s) != 0)
    p = p->proximo;
  if(p == NULL) printf("Nao foi encontrado\n");
  else escrever_aluno(p);
  return p;
}

void elimina(lista * l) { /* elimina um registo da lista ligada */
  aluno *p, *pa; char resp;
  p = busca(l); if(p == NULL) return;
  printf("Deseja remover (s,n)? ");
  scanf(" %c", &resp);
  if(resp =='s')
    if(p == l->primeiro) {
      l->primeiro = p->proximo;
      free(p);
    } else {
      pa = l->primeiro;
      while(pa->proximo != p) pa=pa->proximo;
      pa->proximo = p->proximo;
      free(p);
    }
}
```

Remoção de um elemento de uma lista



```c
void lista(lista * l) {
  aluno *p;
  p = l->primeiro;
  while(p != NULL) {
    escrever_aluno(p);
    p=p->proximo;
  }
}

void menu(lista * l) {
  char op;
  printf("Operacoes: i(nsere), e(limina), b(usca), l(ista), s(ai)\n");
  printf(">> Operacao desejada? ");
  scanf(" %c", &op);
  switch(op) {
    case 'i': insere(l); break;
    case 'e': elimina(l); break;
    case 'b': busca(l); break;
    case 'l': lista(l); break;
    case 's': exit(0);
    default: printf("Operacao nao definida\n");
  }
}

int main() {
  lista *novaLista = init();
  while(1){
    menu(novaLista);
  }
}
```