Algoritmos e estruturas de Dados A07

Listas encadeadas

Prof. Mauri Ferrandin

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

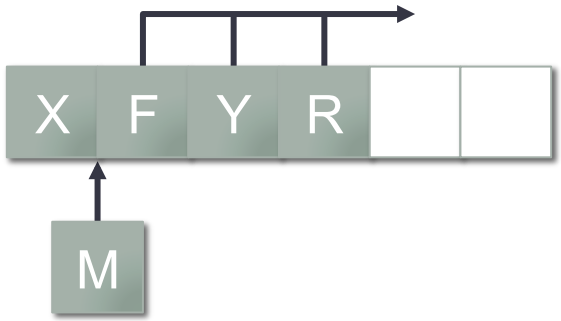
Técnicas de Implementação de Listas

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

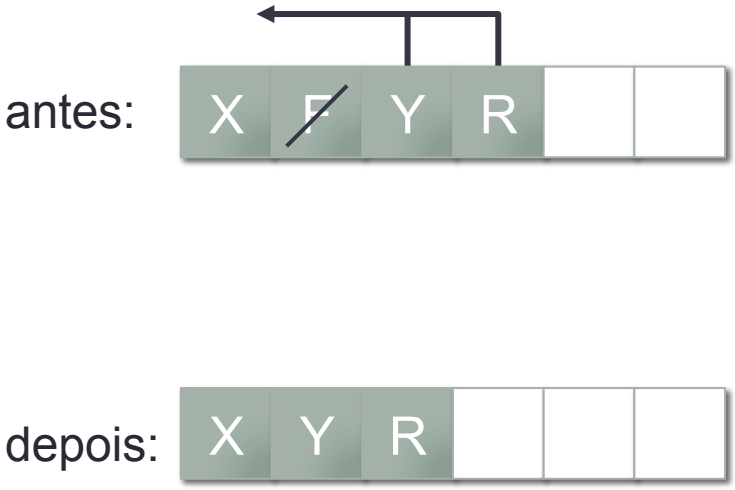
Implementação baseada em Vetores

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 monitorização constante.

Inserção de M:



Remoção de F:



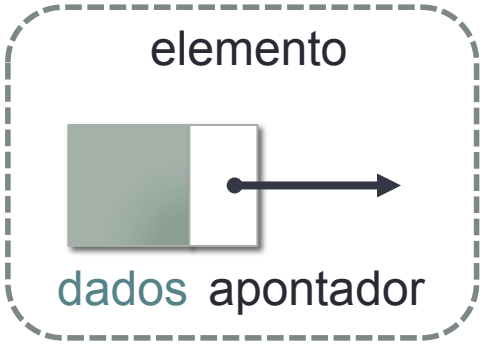
Implementação baseada em Vetores

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

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

}

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

```
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

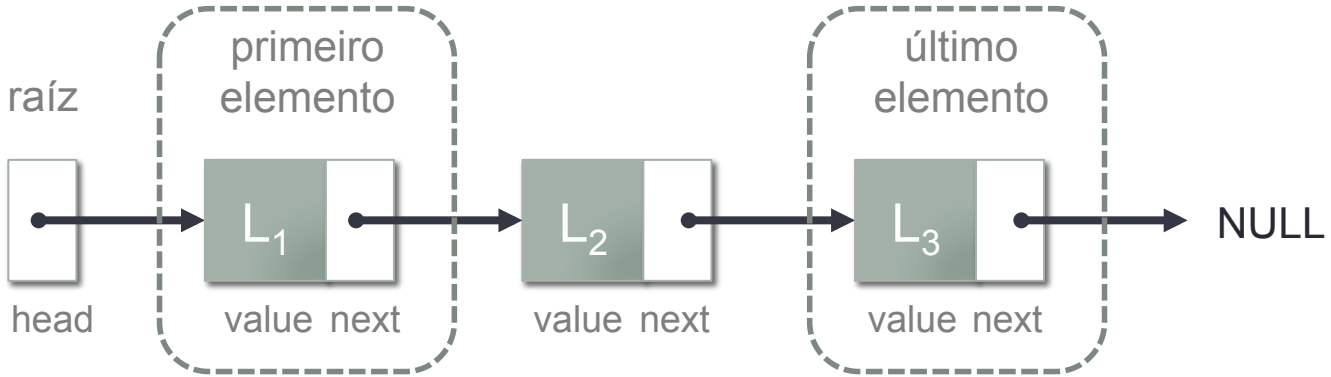
É uma estrutura de dados linear que consiste numa sequência de elementos, em que cada **elemento** inclui um (ou mais) campo(s) com **dados** e um **ponteiro**. O tamanho da lista é facilmente alterado por alocação dinâmica.



- 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.

Lista Ligada ou Encadeada

```
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)
```



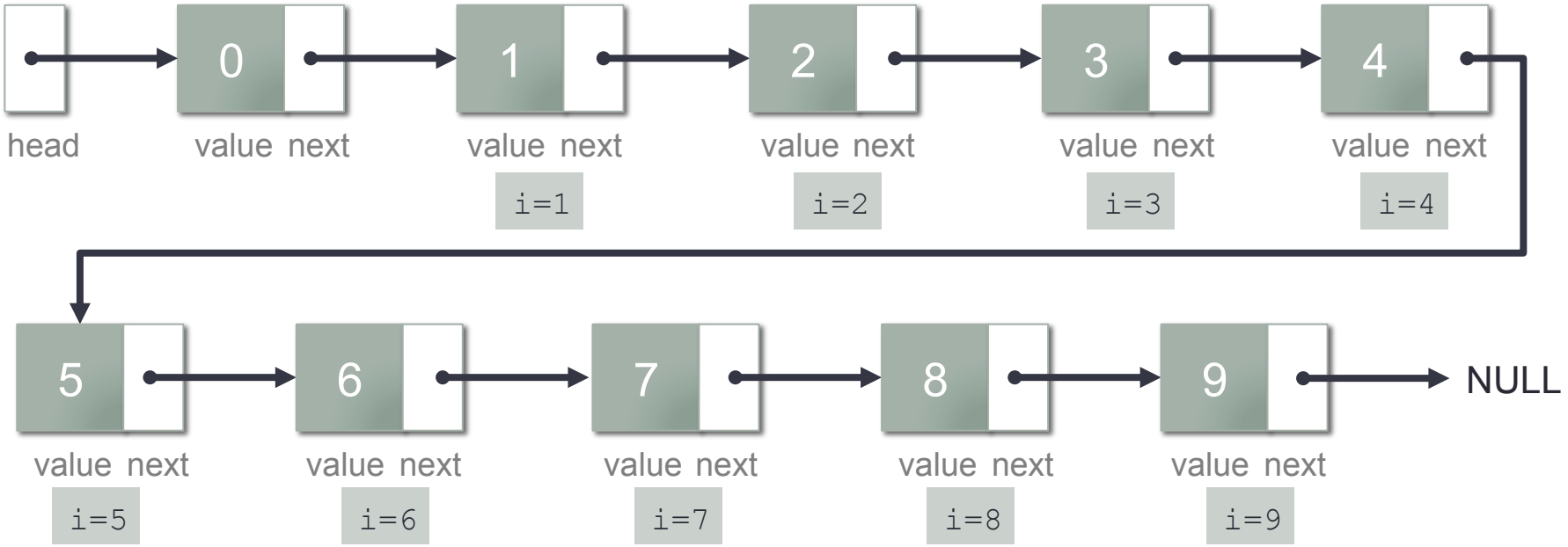
Lista Encadeada (Exemplo)

```
head = newItem(0);

curr = head;
```

Cria e insere o primeiro elemento da lista

```
for (i=1 ; i<10; i++) {
    curr->next = newItem(i);
    curr = curr->next;
}
```

Cria e insere novos elementos no fim da lista



Lista Encadeada (Exemplo)

```c
#include <stdlib.h>
#include <stdio.h>

typedef struct listItem {
  int value;
  struct listItem * next;
} Element;

typedef struct {
  Element *start, *end;
}listHead;

/* adiciona um novo elemento do tipo listItem com o valor inteiro val na lista */
listHead *newItem(listHead * list, int val){
  Element *item = (Element *)malloc(sizeof(Element));
  item->value = val;
  item->next = NULL;
  if (list->start == NULL) {
    list->start = item;
    list->end = item;
  }else{
    list->end->next = item;
    list->end = item;
  }
  return list;
}
```

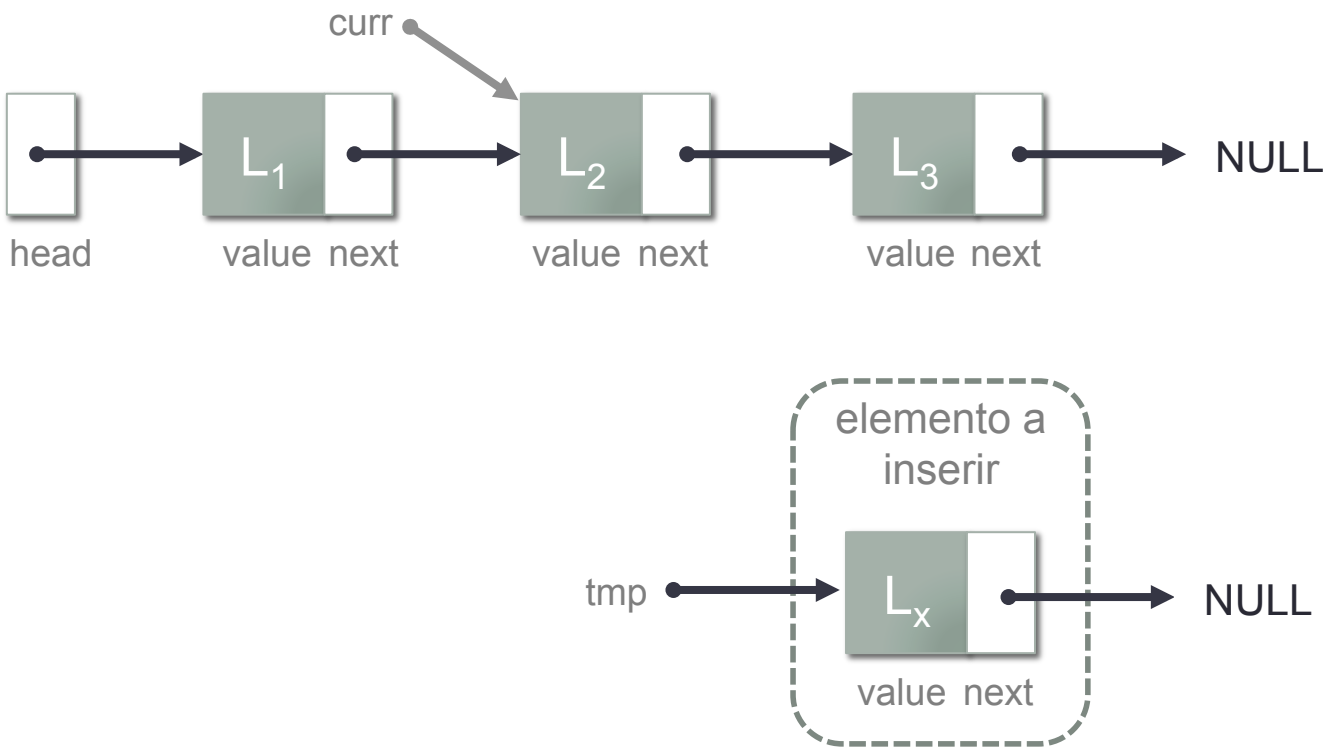
Lista Encadeada (Exemplo cont.)

```
/* percorre e imprime os dados armazenados na lista */
void printList(listHead * list) {
  if (list->start == NULL) {
    printf("Lista vazia!\n");
    return;
  }
  Element *item = list->start;
  printf("Lista:");
  while(item != NULL) { /* percorre todos os elementos da lista */
    printf(" %d", item->value);
    item = item->next ;
  }
  printf("\n");
}

/* inicializa uma lista vazia */
listHead *init(){
  listHead *list = (listHead *)malloc(sizeof(listHead));
  list->start = NULL;
  list->end = NULL;
  return list;
}
```

Lista Encadeada (Exemplo cont.)

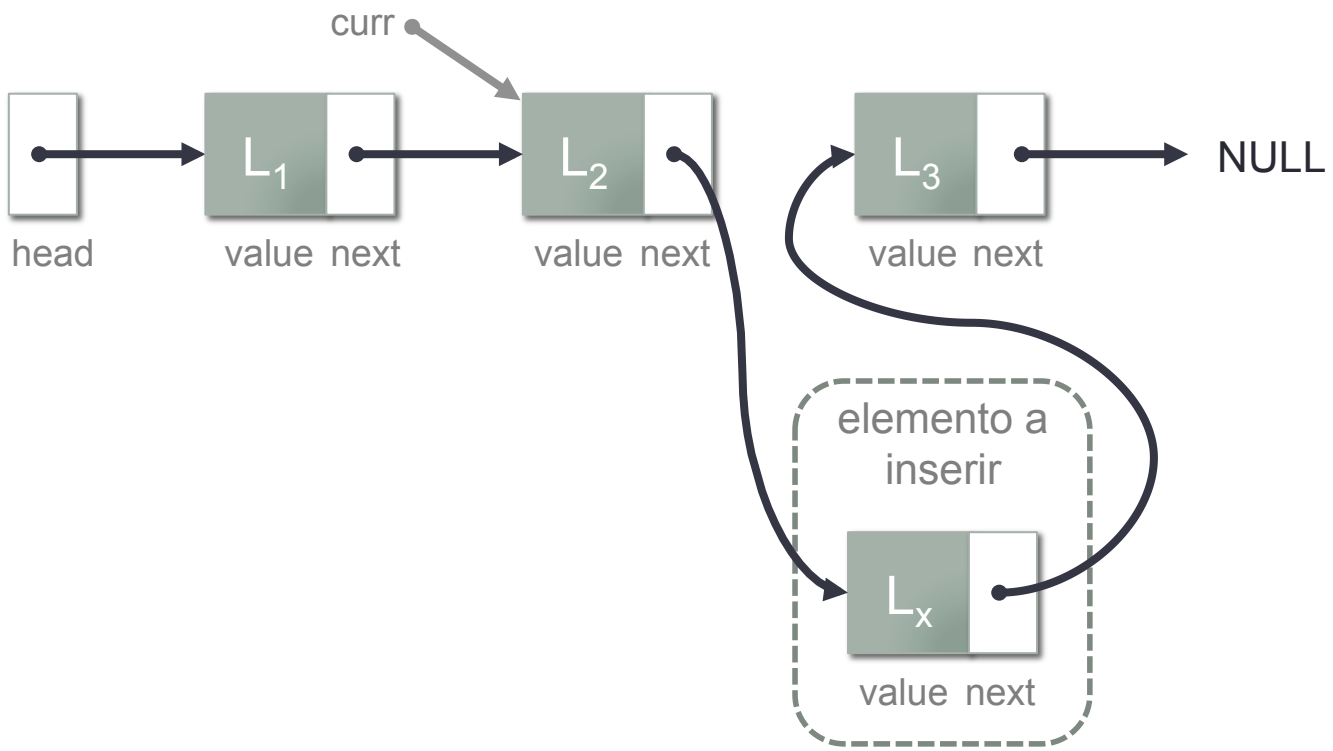
```
int main() {

  int i;

  listHead *newList = init();
  for(i=1 ; i<10; i++) {
      newItem(newList, i);
  }

  printList(newList);
  /* liberar memória! */
}
```

Inserir Elementos na Lista



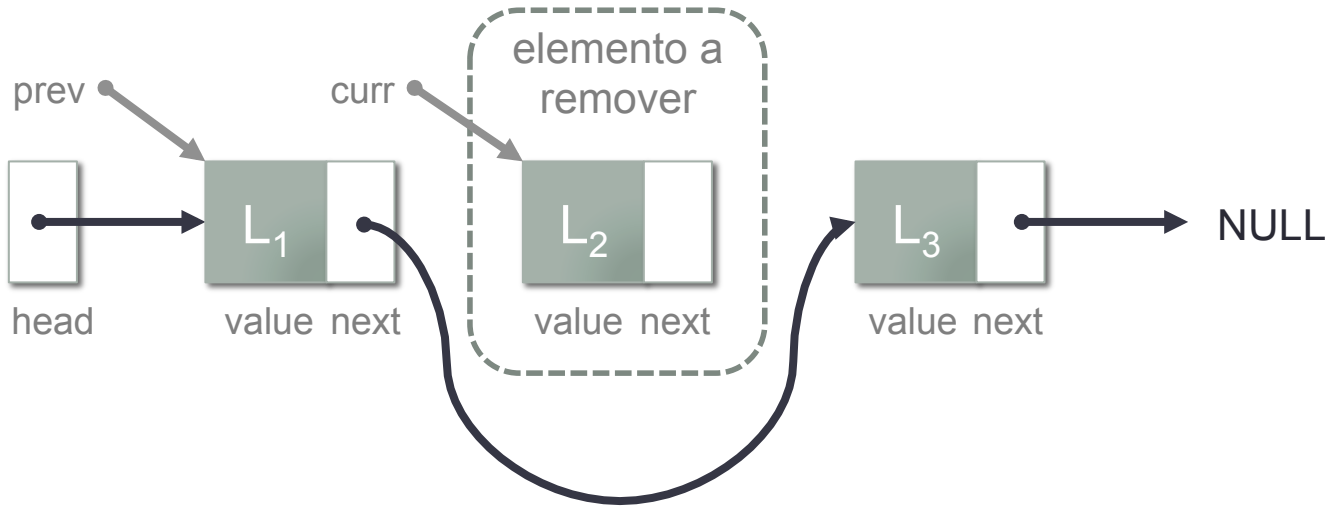
Para inserir um novo elemento na lista, referenciado por um ponteiro temporário `tmp`, é necessário identificar a posição onde esse elemento irá ficar e obter um apontador para o elemento seguinte, i.e. `curr->next`.

Inserir Elementos na Lista



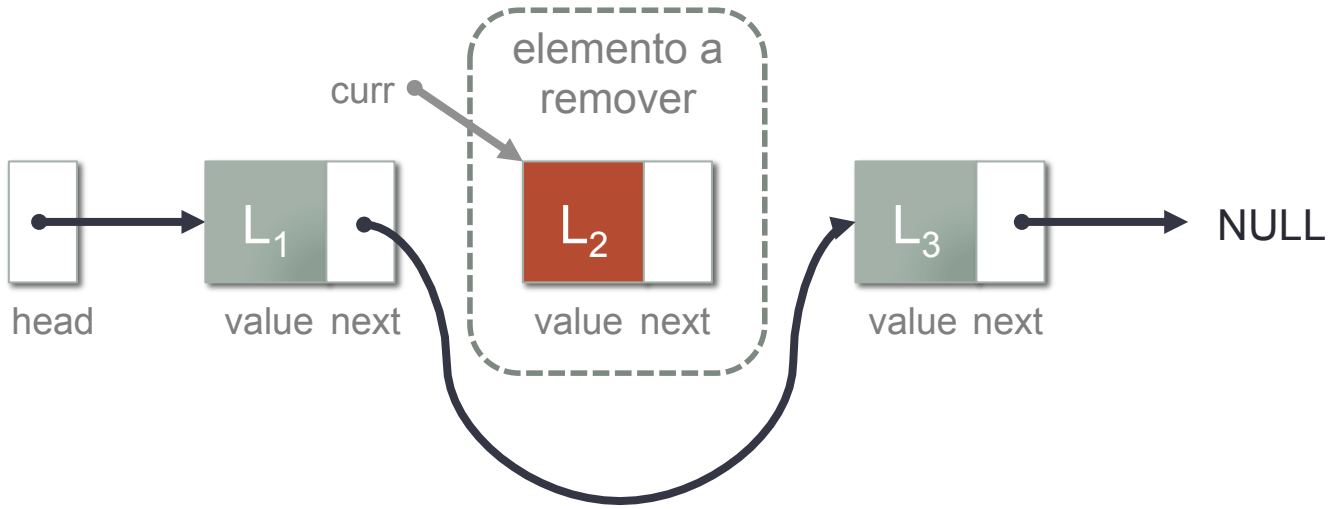
A inserção efetua-se atualizando os apontadores `tmp->next = curr->next` e `curr->next = tmp`.

Remover Elementos da Lista



Depois de identificar o elemento a remover, é necessário atualizar o apontador `prev->next = curr->next` e eliminar o apontador `curr` da memória.

Fugas de Memória



O que acontecerá ao elemento removido, quando não se liberta ou se atribui um novo endereço ao apontador do elemento (curr)?

O elemento fica permanentemente inacessível e não será possível libertar o espaço de memória que tinha sido alocado para o armazenar. Esta “perda” de elementos origina **fugas de memória** (***memory leaks***). Quando as fugas são significativas pode ocorrer falhas no sistema!

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 actual 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

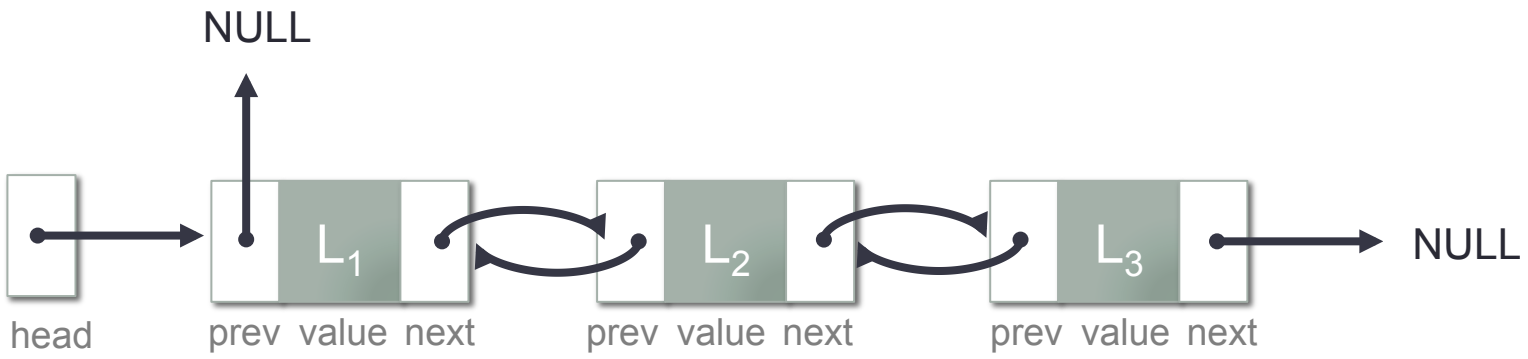
Implementação baseada em Listas Ligadas (desvantagens)

- 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

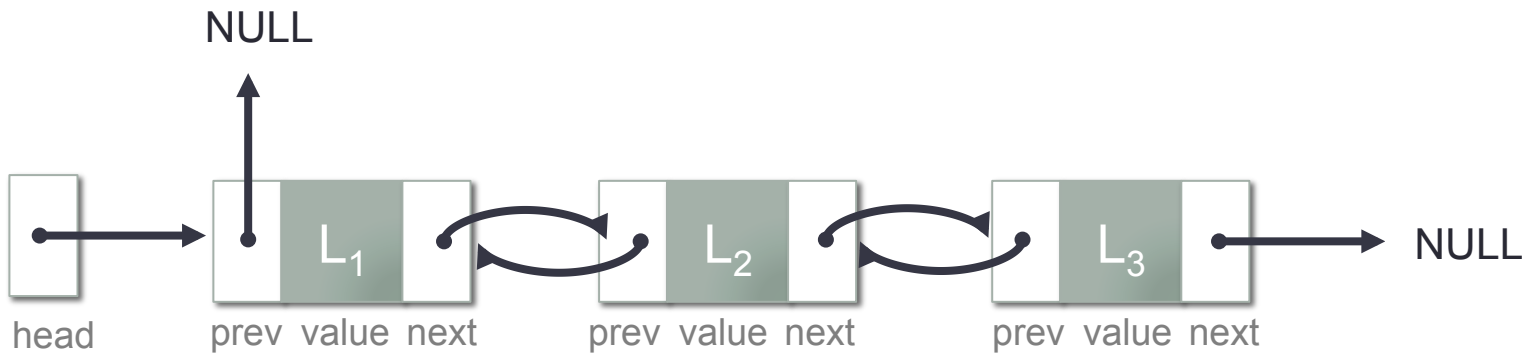
Uma **lista duplamente ligada** é um tipo especial de lista, em que cada elemento inclui dois apontadores, um para o elemento anterior e outro para o elemento seguinte da lista.

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



Lista Duplamente Ligada

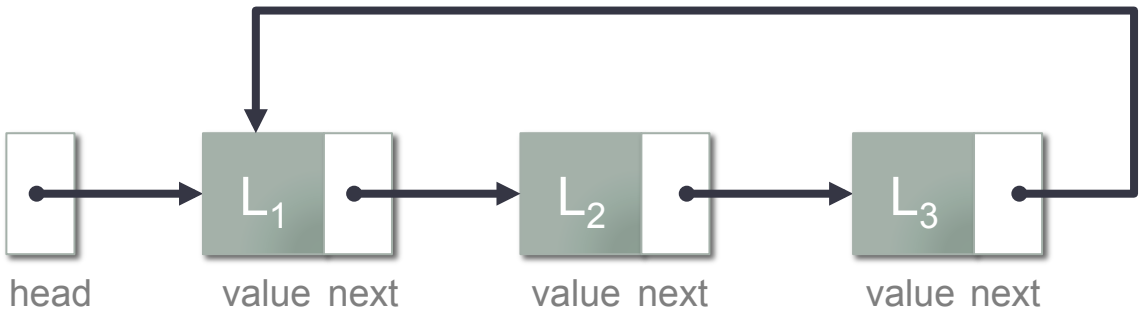
```
struct listItem {
   data_type value;              /* valor do elemento da lista */
   struct listItem *prev;        /* apontador para o elemento anterior */
   struct listItem *next;        /* apontador para o próximo elemento */
};

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



Listas Circulares

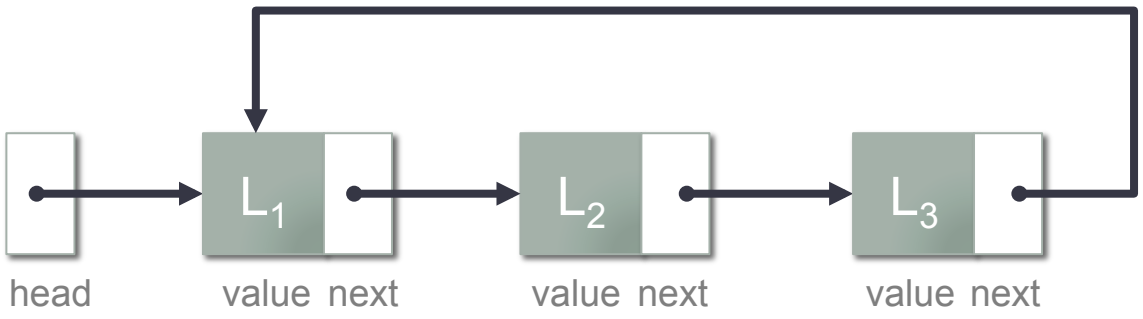
Uma **lista circular** é um tipo especial de lista ligada, em que o último elemento referencia o primeiro elemento da lista.

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.

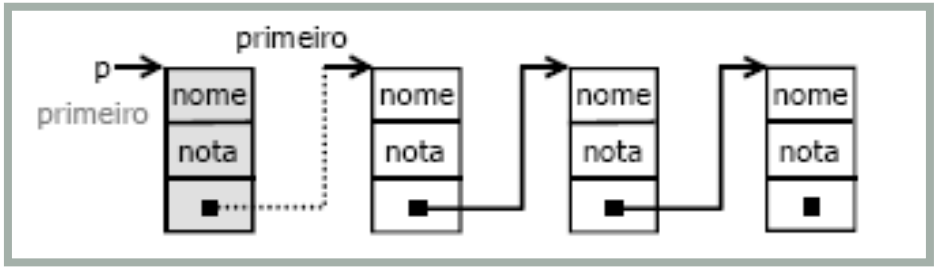
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NCAR 100

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

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


lista* init(){
  lista *l = (lista*)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);
}
```

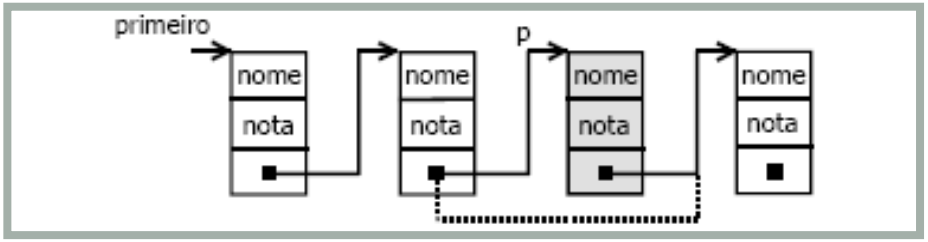
```
aluno* novo_aluno(void) {
  aluno *p;
  p = (aluno*)malloc(sizeof(aluno));
  p->proximo = NULL;
  return p;
}

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) { /* efectua 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 (a cinzento) 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);
  }
}
```