



Listas Circulares – Alocação Encadeada

Listas Circulares

Listas lineares encadeadas possuem alguma deficiência?

Sim.

Qual?

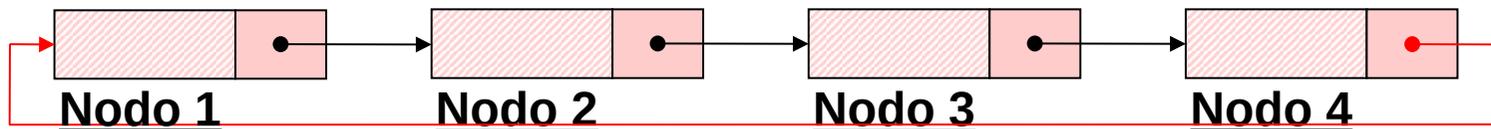
Dado um ponteiro p para um nodo de uma lista linear encadeada, não podemos atingir nenhum nodo que antecede o apontado por p .

Contudo, se fizermos uma pequena alteração na estrutura da lista que temos trabalhado eliminaremos esta restrição.

Listas Circulares

Se fizermos com que o campo *next* do último nodo, ao invés de conter **NULL**, armazenar o endereço do primeiro nodo da lista, resolveremos este problema.

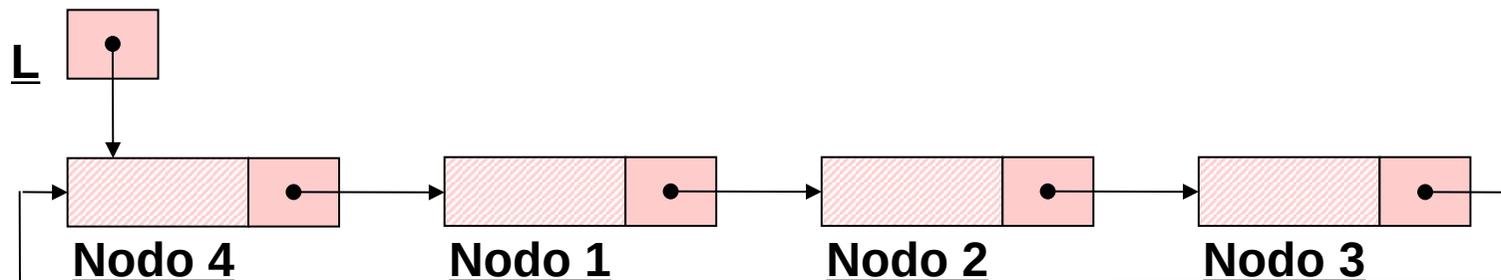
Este tipo de lista é denominado lista circular, possuindo a forma abaixo:



Listas Circulares

Uma pergunta surge: para qual elemento apontar para que se tenha uma referência a lista circular?

Uma convenção útil é fazer com que o ponteiro externo para a lista circular aponte para o último elemento.



Listas Circulares

Como veremos a definição do TAD LISTA_CIRCULAR é praticamente a mesma do TAD LISTA_ENC, apenas algumas pequenas alterações são necessárias nas implementações de algumas operações.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_CIRCULAR;
7  void cria_lista (LISTA_CIRCULAR *);
8  int eh_vazia (LISTA_CIRCULAR);
9  int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR);
```

Listas Circulares

Com base no que foi visto implemente a operação `cria_lista()` que compõem o TAD `LISTA_CIRCULAR`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_CIRCULAR;
7  void cria_lista (LISTA_CIRCULAR *);
8  int eh_vazia (LISTA_CIRCULAR);
9  int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR) ;
```

```
1 void cria_lista (LISTA_CIRCULAR *p1)
2 {
3     *p1=NULL;
4 }
```

Listas Circulares

Com base no que foi visto implemente a operação `eh_vazia()` que compõem o TAD `LISTA_CIRCULAR`.

```
1 typedef struct nodo
2 {
3     int inf;
4     struct nodo * next;
5 }NODO;
6 typedef NODO * LISTA_CIRCULAR;
7 void cria_lista (LISTA_CIRCULAR *);
8 int eh_vazia (LISTA_CIRCULAR);
9 int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR) ;
```

```
1  int eh_vazia (LISTA_CIRCULAR l)
2  {
3      return (!l);
4  }
```

Listas Circulares

Com base no que foi visto implemente a operação tam() que compõem o TAD LISTA_CIRCULAR.

```
1 typedef struct nodo
2 {
3     int inf;
4     struct nodo * next;
5 }NODO;
6 typedef NODO * LISTA_CIRCULAR;
7 void cria_lista (LISTA_CIRCULAR *);
8 int eh_vazia (LISTA_CIRCULAR);
9 int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR) ;
```

```
1  int tam (LISTA_CIRCULAR l)
2  {
3      if (!l)
4          return (0);
5      else
6      {
7          LISTA_CIRCULAR aux; /* NODO *aux; */
8          int cont;
9          for (cont=1, aux=l->next; aux!=l ; cont++)
10             aux = aux->next;
11             return (cont);
12     }
13 }
```

Listas Circulares

Com base no que foi visto implemente a operação `ins()` que compõem o TAD `LISTA_CIRCULAR`.

```
1 typedef struct nodo
2 {
3     int inf;
4     struct nodo * next;
5 }NODO;
6 typedef NODO * LISTA_CIRCULAR;
7 void cria_lista (LISTA_CIRCULAR *);
8 int eh_vazia (LISTA_CIRCULAR);
9 int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR) ;
```

Listas Circulares

Dicas:

A posição é válida?

Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

```
1 void ins (LISTA_CIRCULAR *pl, int v, int k) {
2     NODO *novo;
3     int tamanho = tam(*pl);
4     if (k < 1 || k > tamanho+1) {
5         printf ("\nERRO! Posição invalida para insercao.\n");
6         exit (1);
7     }
8     novo = (NODO *) malloc (sizeof(NODO));
9     if (!novo) {
10        printf ("\nERRO! Memoria insuficiente!\n");
11        exit (2);
12    }
13    novo->inf = v;
```

```
14     if (*p1==NULL)    {
15         novo->next=novo;
16         *p1 = novo;
17     } else {
18         LISTA_CIRCULAR aux=*p1;
19         if (k==tamanho+1)
20             *p1=novo;
21         else
22             for (; k>1; aux=aux->next, k--);
23         novo->next = aux->next;
24         aux->next = novo;
25     }
26 }
```

Listas Circulares

Com base no que foi visto implemente a operação `recup()` que compõem o TAD `LISTA_CIRCULAR`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_CIRCULAR;
7  void cria_lista (LISTA_CIRCULAR *);
8  int eh_vazia (LISTA_CIRCULAR);
9  int tam (LISTA_CIRCULAR);
10 void ins (LISTA_CIRCULAR *, int, int);
11 int recup (LISTA_CIRCULAR, int);
12 void ret (LISTA_CIRCULAR *, int);
13 void destruir (LISTA_CIRCULAR) ;
```