BCC202 - Estruturas de Dados I

Aula 11: Listas (Parte II)

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

Conteúdo

Introdução

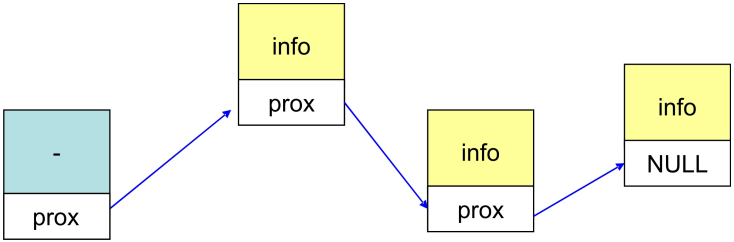
Sobre Listas

- **Na última aula** (Parte 1):
  - Dois tipos de listas: implementadas por ARRAY e implementadas por PONTEIRO.
  - Exemplos de implementação de listas por ARRAY e operações relacionadas à implementação por PONTEIRO.
  - **Implementação por PONTEIRO** também é denominada de **Lista Encadeada** (ou **Lista Simplesmente Encadeada**).
- **Na aula de hoje** (Parte 2):
  - Três variações de Lista Encadeada: **COM Cabeça**, **SEM cabeça** e **Duplamente Encadeada**.
  - Exemplos de código de listas COM e SEM cabeça.
  - Conceitos de lista Duplamente Encadeada.

Implementação por PONTEIRO

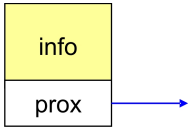
Lista Encadeada COM cabeça: Características

- Tamanho da lista não é pré-definido.
- Cada elemento (célula) guarda quem é o próximo.
- Elementos não estão contíguos na memória.



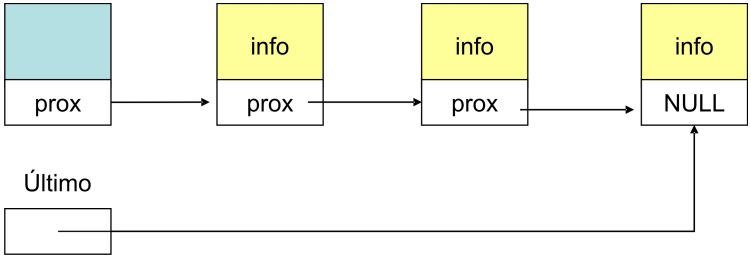
Lista Encadeada COM cabeça: Elementos

- **Elemento da lista (Célula)**:
  - Armazena as informações sobre cada elemento.
  - Aponta para o próximo elemento.
- Assim, é definido como uma estrutura que possui:
  - Campos representando as informações do elemento.
  - Ponteiro para o próximo elemento.



Lista Encadeada COM cabeça: Elementos

- Uma lista pode ter uma célula **cabeça**, antecedendo o primeiro elemento.
- Pode possuir também um apontador para o **último** elemento.



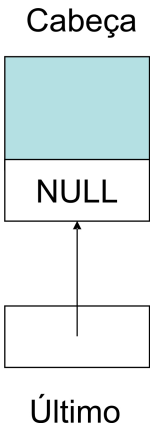
Lista Encadeada COM cabeça: Estruturas básicas

```
1  typedef struct {
2    /* Componentes de um item: "info" */
3  } Item;
4
5  typedef struct cel {
6    struct cel *pProx;
7    Item item;
8  } Celula;
9
10 typedef struct {
11   Celula *pPrimeiro, *pUltimo;
12 } Lista;
```

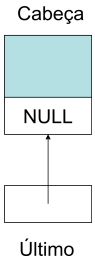
Lista Encadeada COM cabeça: Operações básicas

```
14 /* Procedimentos e funções basicas do TAD */
15 void ListaInicia(Lista *pLista);
16 int ListaEhVazia(Lista *pLista);
17 int ListaInsere(Lista *pLista, Item x);
18 int ListaRetiraPrimeiro(Lista *pLista, Item *pX);
19 void ListaImprime(Lista *pLista);
```

Lista Encadeada COM cabeça: Criar Lista Vazia

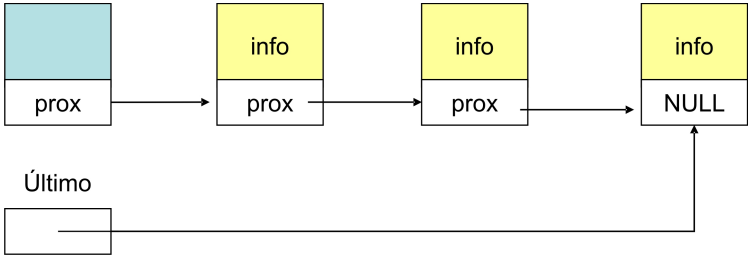


Lista Encadeada COM cabeça: Lista Vazia



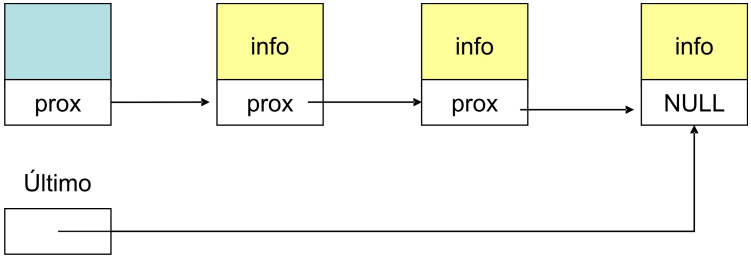

```
21 /* Inicia as variáveis da lista */
22 void ListaInicia(Lista *pLista) {
23   pLista->pPrimeiro = (Celula*) malloc(sizeof(Celula));
24   pLista->pUltimo = pLista->pPrimeiro;
25   pLista->pPrimeiro->pProx = NULL;
26 }
27
28 /* Retorna se a lista é vazia */
29 int ListaEhVazia(Lista *pLista) {
30   return (pLista->pPrimeiro == pLista->pUltimo);
31 }
```

Lista Encadeada COM cabeça: INSERÇÃO de Novos Elementos



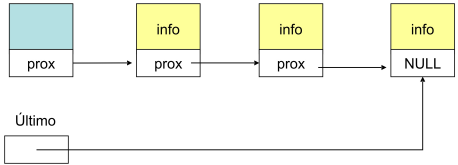
- 3 opções de posição onde se pode inserir:
  - Primeira posição.
  - Última posição.
  - Após um elemento `E`.

Lista Encadeada COM cabeça: INSERÇÃO de Novos Elementos



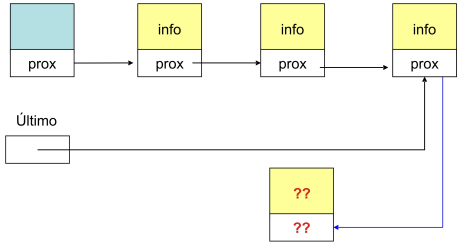
- 3 opções de posição onde se pode inserir:
  - Primeira posição.
  - Última posição.
  - Após um elemento `E`.

Lista Encadeada COM cabeça: INSERÇÃO na Última Posição



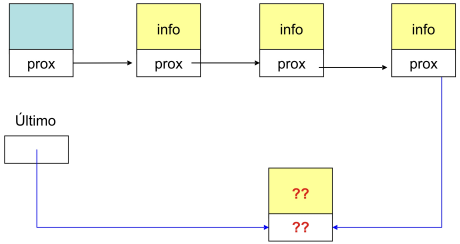

```
33 /* Insere um item no final da lista */
34 int ListaInsere(Lista *pLista, Item x) {
35   pLista->pUltimo->pProx=(Celula*)malloc(sizeof(Celula));
36   pLista->pUltimo = pLista->pUltimo->pProx;
37   pLista->pUltimo->Item = x;
38   pLista->pUltimo->pProx = NULL;
39   return 1;
40 }
```

Lista Encadeada COM cabeça: INSERÇÃO na Última Posição



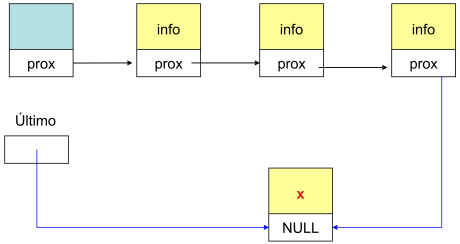

```
33 /* Insere um item no final da lista */
34 int ListaInsere(Lista *pLista, Item x) {
35   pLista->pUltimo->pProx=(Celula*)malloc(sizeof(Celula));
36   pLista->pUltimo = pLista->pUltimo->pProx;
37   pLista->pUltimo->Item = x;
38   pLista->pUltimo->pProx = NULL;
39   return 1;
40 }
```

Lista Encadeada COM cabeça: INSERÇÃO na Última Posição



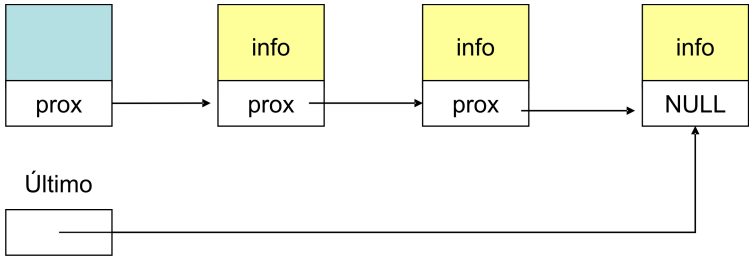

```
33 /* Insere um item no final da lista */
34 int ListaInsere(Lista *pLista, Item x) {
35   pLista->pUltimo->pProx=(Celula*)malloc(sizeof(Celula));
36   pLista->pUltimo = pLista->pUltimo->pProx;
37   pLista->pUltimo->Item = x;
38   pLista->pUltimo->pProx = NULL;
39   return 1;
40 }
```

Lista Encadeada COM cabeça: INSERÇÃO na Última Posição



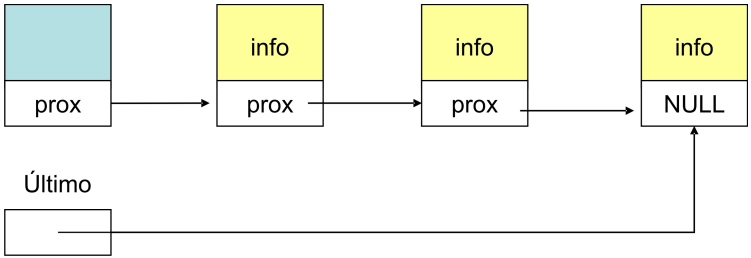

```
33 /* Insere um item no final da lista */
34 int ListaInsere(Lista *pLista, Item x) {
35   pLista->pUltimo->pProx=(Celula*)malloc(sizeof(Celula));
36   pLista->pUltimo = pLista->pUltimo->pProx;
37   pLista->pUltimo->Item = x;
38   pLista->pUltimo->pProx = NULL;
39   return 1;
40 }
```

Lista Encadeada COM cabeça: RETIRADA de Elementos



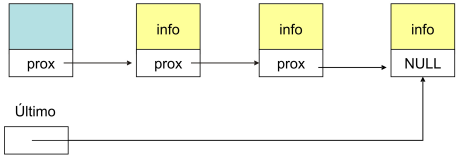
- 3 opções de posição onde se pode retirar:
  - Primeira posição.
  - Última posição.
  - Um elemento E.

Lista Encadeada COM cabeça: RETIRADA de Elementos



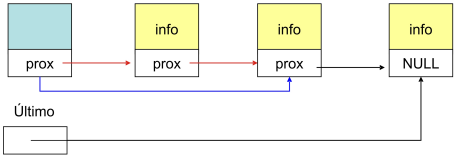
- 3 opções de posição onde se pode retirar:
  - Primeira posição.
  - Última posição.
  - Um elemento E.

Lista Encadeada COM cabeça: RETIRADA na Primeira Posição



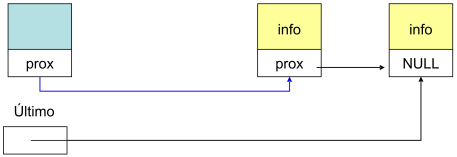

```
41 /* Retira o primeiro item da lista, retornando-o em *px */
42 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
43   if (ListaEhVazia(pLista))
44     return 0;
45   Celula *pAux;
46   pAux = pLista->pPrimeiro->pProx;
47   *pX = pAux->item;
48   pLista->pPrimeiro->pProx = pAux->pProx;
49   free(pAux);
50   return 1;
51 }
```

Lista Encadeada COM cabeça: RETIRADA na Primeira Posição




```
41 /* Retira o primeiro item da lista, retornando-o em *px */
42 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
43   if (ListaEhVazia(pLista))
44     return 0;
45   Celula *pAux;
46   pAux = pLista->pPrimeiro->pProx;
47   *pX = pAux->item;
48   pLista->pPrimeiro->pProx = pAux->pProx;
49   free(pAux);
50   return 1;
51 }
```

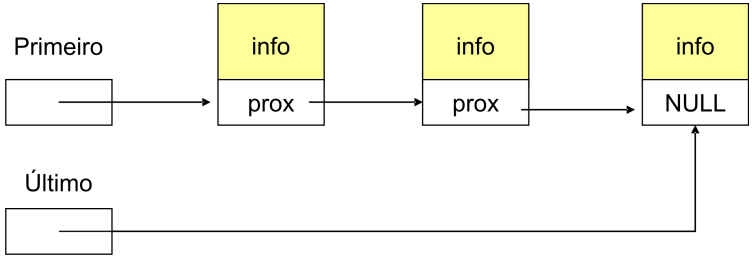
Lista Encadeada COM cabeça: RETIRADA na Primeira Posição




```
41 /* Retira o primeiro item da lista, retornando-o em *px */
42 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
43   if (ListaEhVazia(pLista))
44     return 0;
45   Celula *pAux;
46   pAux = pLista->pPrimeiro->pProx;
47   *pX = pAux->item;
48   pLista->pPrimeiro->pProx = pAux->pProx;
49   free(pAux);
50   return 1;
51 }
```

Lista Encadeada SEM cabeça: Características

- Se diferencia da lista COM cabeça apenas pelo fato de que não possui a célula **cabeça**, mas apenas um apontador para o primeiro elemento.



Lista Encadeada SEM cabeça: Estruturas básicas

```
1  typedef struct {
2    /* Componentes de um item: "info" */
3  } Item;
4
5  typedef struct cel {
6    struct cel *pProx;
7    Item item;
8  } Celula;
9
10 typedef struct {
11   Celula *pPrimeiro, *pUltimo;
12 } Lista;
```

Lista Encadeada SEM cabeça: Operações básicas

```
14 /* Procedimentos e funções basicas do TAD */
15 void ListaInicia(Lista *pLista);
16 int ListaEhVazia(Lista *pLista);
17 int ListaInsere(Lista *pLista, Item x);
18 int ListaRetiraPrimeiro(Lista *pLista, Item *pX);
19 void ListaImprime(Lista *pLista);
```

Lista Encadeada SEM cabeça: Lista Vazia

```
21 /* Inicia as variáveis da lista */
22 void ListaInicia(Lista *pLista) {
23   pLista->pPrimeiro = NULL;
24   pLista->pUltimo = NULL;
25 }
26
27 /* Retorna se a lista é vazia */
28 int ListaEhVazia(Lista *pLista) {
29   return (pLista->pPrimeiro == NULL);
30 }
```

Na lista COM Cabeça tínhamos um código ligeiramente diferente:

```
21 /* Inicia as variáveis da lista */
22 void ListaInicia(Lista *pLista) {
23   pLista->pPrimeiro = (Celula*) malloc(sizeof(Celula));
24   pLista->pUltimo = pLista->pPrimeiro;
25   pLista->pPrimeiro->pProx = NULL;
26 }
27
28 /* Retorna se a lista é vazia */
29 int ListaEhVazia(Lista *pLista) {
30   return (pLista->pPrimeiro == pLista->pUltimo);
31 }
```

Lista Encadeada SEM cabeça: Lista Vazia

```
21 /* Inicia as variáveis da lista */
22 void ListaInicia(Lista *pLista) {
23   pLista->pPrimeiro = NULL;
24   pLista->pUltimo = NULL;
25 }
26
27 /* Retorna se a lista é vazia */
28 int ListaEhVazia(Lista *pLista) {
29   return (pLista->pPrimeiro == NULL);
30 }
```

Na lista COM Cabeça tínhamos um código ligeiramente diferente:

```
21 /* Inicia as variáveis da lista */
22 void ListaInicia(Lista *pLista) {
23   pLista->pPrimeiro = (Celula*) malloc(sizeof(Celula));
24   pLista->pUltimo = pLista->pPrimeiro;
25   pLista->pPrimeiro->pProx = NULL;
26 }
27
28 /* Retorna se a lista é vazia */
29 int ListaEhVazia(Lista *pLista) {
30   return (pLista->pPrimeiro == pLista->pUltimo);
31 }
```

Lista Encadeada SEM cabeça: INSERÇÃO na Última Posição

```
32 /* Insere um item no final da lista */
33 int ListaInsere(Lista *pLista, Item x) {
34   Celula *novo = (Celula*)malloc(sizeof(Celula));
35   novo->Item = x;
36   novo->pProx = NULL;
37   if (ListaEhVazia(pLista)) {
38     pLista->pPrimeiro = novo;
39     pLista->pUltimo = novo;
40   } else {
41     pLista->pUltimo->pProx = novo;
42     pLista->pUltimo = novo;
43   }
44 }
```

Na lista COM Cabeça tínhamos um código muito diferente (mais simples):

```
33 /* Insere um item no final da lista */
34 int ListaInsere(Lista *pLista, Item x) {
35   pLista->pUltimo->pProx=(Celula*)malloc(sizeof(Celula));
36   pLista->pUltimo = pLista->pUltimo->pProx;
37   pLista->pUltimo->Item = x;
38   pLista->pUltimo->pProx = NULL;
39 }
```

Lista Encadeada SEM cabeça: INSERÇÃO na Última Posição

```
32 /* Insere um item no final da lista */
33 int ListaInsere(Lista *pLista, Item x) {
34   Celula *novo = (Celula*)malloc(sizeof(Celula));
35   novo->Item = x;
36   novo->pProx = NULL;
37   if (ListaEhVazia(pLista)) {
38     pLista->pPrimeiro = novo;
39     pLista->pUltimo = novo;
40   } else {
41     pLista->pUltimo->pProx = novo;
42     pLista->pUltimo = novo;
43   }
44 }
```

Na lista COM Cabeça tínhamos um código muito diferente (mais simples):

```
33 /* Insere um item no final da lista */
34 int ListaInsere(Lista *pLista, Item x) {
35   pLista->pUltimo->pProx=(Celula*)malloc(sizeof(Celula));
36   pLista->pUltimo = pLista->pUltimo->pProx;
37   pLista->pUltimo->Item = x;
38   pLista->pUltimo->pProx = NULL;
39 }
```

Lista Encadeada SEM cabeça: RETIRADA na Primeira Posição

```
40 /* Retira o primeiro item da lista, retornando-o em *px */
41 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
42   if (ListaEhVazia(pLista))
43     return 0;
44   Celula *pAux;
45   pAux = pLista->pPrimeiro->pProx;
46   *pX = pAux->item;
47   pLista->pPrimeiro->pProx = pAux->pProx;
48   free(pAux);
49   return 1;
50 }
```

Na lista COM Cabeça tínhamos um código exatamente igual:

```
40 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
41   if (ListaEhVazia(pLista))
42     return 0;
43   Celula *pAux;
44   pAux = pLista->pPrimeiro->pProx;
45   *pX = pAux->item;
46   pLista->pPrimeiro->pProx = pAux->pProx;
47   free(pAux);
48   return 1;
49 }
```

Lista Encadeada SEM cabeça: RETIRADA na Primeira Posição

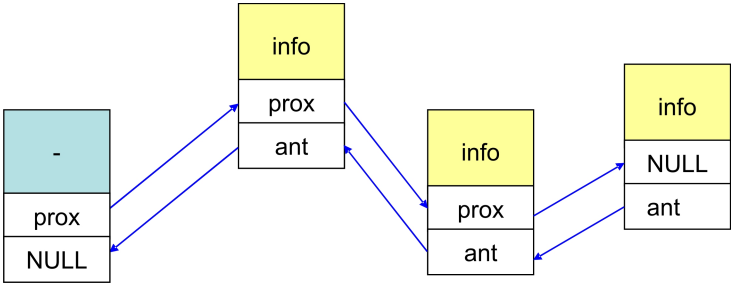
```
40 /* Retira o primeiro item da lista, retornando-o em *px */
41 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
42   if (ListaEhVazia(pLista))
43     return 0;
44   Celula *pAux;
45   pAux = pLista->pPrimeiro->pProx;
46   *pX = pAux->item;
47   pLista->pPrimeiro->pProx = pAux->pProx;
48   free(pAux);
49   return 1;
50 }
```

Na lista COM Cabeça tínhamos um código exatamente igual:

```
40 int ListaRetiraPrimeiro(Lista *pLista, Item *pX) {
41   if (ListaEhVazia(pLista))
42     return 0;
43   Celula *pAux;
44   pAux = pLista->pPrimeiro->pProx;
45   *pX = pAux->item;
46   pLista->pPrimeiro->pProx = pAux->pProx;
47   free(pAux);
48   return 1;
49 }
```

Lista Duplamente Encadeada: Características

- Se diferencia das anteriores pelo fato de que cada elemento aponta para os elementos anterior e posterior a ele.



- Muito útil quando ocorrem muitas inserções e remoções, principalmente de elementos intermediários.

Lista Duplamente Encadeada: Variações

- Existem muitas variações da lista duplamente encadeada, muitas delas servem também para a lista encadeada. Alguns exemplos:
  - **Com sentinelas**: possui dois elementos especiais, que não armazenam dados, os **sentinelas**, a **cabeça** da lista (***head***) e a **cauda** (***tail***). O elemento anterior da cabeça aponta sempre para NULL enquanto que no nó cauda quem aponta para NULL é próximo.
  - **Circular**: conhecida como circular pois o primeiro elemento aponta para o último e vice-versa, formando assim um círculo lógico. pode ser implementado com sentinela ou não.

Conclusão

Conclusão

- Nesta aula tivemos contato com novos exemplos de implementação da estrutura de dados **Lista**.
- O entendimento dos conceitos e das implementações apresentadas nas duas aulas sobre este tópico são muito importantes para o entendimento das estruturas de dados que estão por vir.
- *Próxima aula*: Pilhas.
- Dúvidas?

Exercícios

Exercício 01

- Implemente um TAD Lista Duplamente Encadeada considerando as mesmas operações implementadas para as listas encadeadas.