

AULA 07

ESTRUTURA DE DADOS

Lista ligada circular com nó cabeça

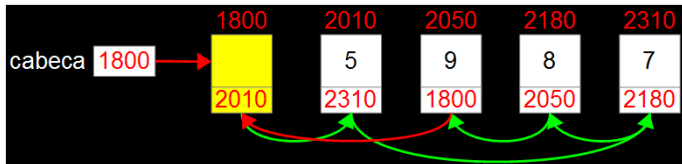
Norton T. Roman & Luciano A. Digiampietri

Lista ligada

Na última aula aprendemos como modelar e gerenciar **listas ligadas** utilizando o que chamamos de **implementação dinâmica**.

- Vantagens: não precisamos **gastar memória** que não estamos usando e não precisamos gerenciar uma lista de **elementos disponíveis**.
- **Hoje** adicionaremos duas características a esta estrutura: ela será **circular** (o último elemento apontará para o primeiro) e possuirá um **nó cabeça** (um elemento inicial que sempre encabeçará a lista).

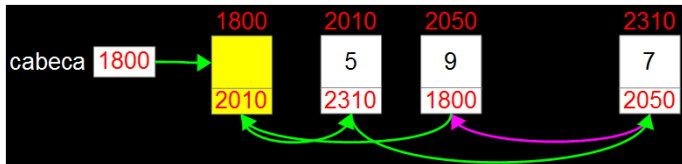
Lista ligada circular com nó cabeça



Temos um ponteiro para o **nó cabeça**

Cada elemento indica seu sucessor e o **último aponta para o cabeça**

Lista ligada circular com nó cabeça

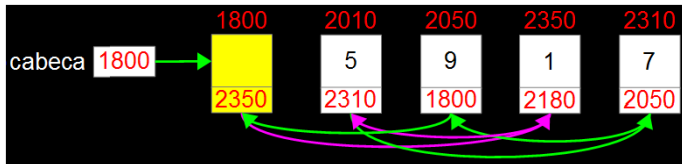


Temos um ponteiro para o **nó cabeça**

Cada elemento indica seu sucessor e o **último aponta para o cabeça**

Como excluimos o elemento 8?

Lista ligada circular com nó cabeça



Temos um ponteiro para o **nó cabeça**

Cada elemento indica seu sucessor e o **último aponta para o cabeça**

Como excluimos o elemento 8?

Como inserimos o elemento 1?

Modelagem

```
#include <stdio.h>
#include <malloc.h>

typedef int bool;
typedef int TIPOCHAVE;

typedef struct {
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct tempRegistro {
    REGISTRO reg;
    struct tempRegistro* prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
    PONT cabeca;
} LISTA;
```

Funções de gerenciamento

Implementaremos funções para:

- Inicializar a estrutura

- Retornar a quantidade de elementos válidos

- Exibir os elementos da estrutura

- Buscar por um elemento na estrutura

- Inserir elementos na estrutura

- Excluir elementos da estrutura

- Reinicializar a estrutura

Inicialização

Para inicializarmos nossa lista ligada circular e com nó cabeça, precisamos:

- Criar o nó cabeça;
- A variável *cabeça* precisa apontar para ele;
- E o nó cabeça apontará para ele mesmo como próximo.

Inicialização

```
void inicializarLista(LISTA* l) {  
    l->cabeca = (PONT) malloc(sizeof(ELEMENTO));  
    l->cabeca->prox = l->cabeca;  
}
```

4200

cabeca ?

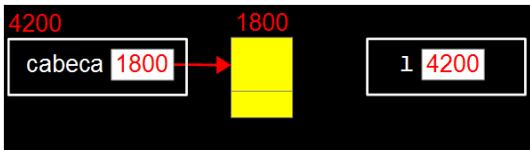
Inicialização

```
void inicializarLista(LISTA* l) {  
    l->cabeca = (PONT) malloc(sizeof(ELEMENTO));  
    l->cabeca->prox = l->cabeca;  
}
```



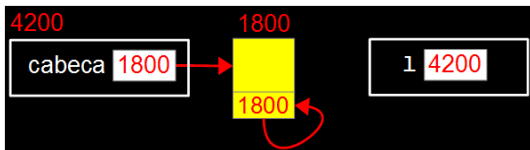
Inicialização

```
void inicializarLista(LISTA* l) {  
    l->cabeca = (PONT) malloc(sizeof(ELEMENTO));  
    l->cabeca->prox = l->cabeca;  
}
```



Inicialização

```
void inicializarLista(LISTA* l) {  
    l->cabeca = (PONT) malloc(sizeof(ELEMENTO));  
    l->cabeca->prox = l->cabeca;  
}
```



Retornar número de elementos

Já que optamos por não criar um campo com o número de elementos na lista, precisaremos **percorrer todos os elementos** para contar quantos são.

```
int tamanho(LISTA* l) {  
    PONT end = l->cabeca->prox;  
    int tam = 0;  
    while (end != l->cabeca) {  
        tam++;  
        end = end->prox;  
    }  
    return tam;  
}
```

Exibição/Impressão

Para exibir os elementos da estrutura precisaremos iterar pelos **elementos** válidos e, por exemplo, **imprimir suas chaves**.

Precisamos lembrar que o nó cabeça não é um dos elementos **válidos** da nossa lista.

Exibição/Impressão

```
void exibirLista(LISTA* l) {  
    PONT end = l->cabeca->prox;  
    printf("Lista: \" ");  
    while (end != l->cabeca) {  
        printf("%i ", end->reg.chave);  
        end = end->prox;  
    }  
    printf("\\n\\n");  
}
```

Buscar por elemento

A função de busca deverá:

- Receber uma chave do usuário

- Retornar o endereço em que este elemento se encontra (caso seja encontrado)

- Retornar *NULL* caso não haja um registro com essa chave na lista

Podemos usar o nó cabeça como sentinela

Busca sequencial

```
PONT buscaSentinela(LISTA* l, TIPOCHAVE ch) {  
    PONT pos = l->cabeca->prox;  
    l->cabeca->reg.chave = ch;  
    while (pos->reg.chave != ch) pos = pos->prox;  
    if (pos != l->cabeca) return pos;  
    return NULL;  
}
```

Busca sequencial - lista ordenada

```
// lista não precisa estar ordenada pelos valores das chaves
PONT buscaSentinela(LISTA* l, TIPOCHAVE ch) {
    PONT pos = l->cabeca->prox;
    l->cabeca->reg.chave = ch;
    while (pos->reg.chave != ch) pos = pos->prox;
    if (pos != l->cabeca) return pos;
    return NULL;
}
```

```
// lista ordenada pelos valores das chaves dos registros
PONT buscaSentinelaOrd(LISTA* l, TIPOCHAVE ch) {
    PONT pos = l->cabeca->prox;
    l->cabeca->reg.chave = ch;
    while (pos->reg.chave < ch) pos = pos->prox;
    if (pos != l->cabeca && pos->reg.chave==ch) return pos;
    return NULL;
}
```

Inserção de um elemento

O usuário passa como parâmetro um registro a ser inserido na lista

Realizaremos a inserção **ordenada pelo valor da chave** do registro passado e não permitiremos a inserção de **elementos repetidos**; Na inserção precisamos identificar **entre quais elementos** o novo elemento será inserido;

Alocaremos memória para o novo elemento.

Precisamos saber quem será o **predecessor do elemento**.

Busca - auxiliar

```
PONT buscaSeqExc(LISTA* l, TIPOCHAVE ch, PONT* ant) {  
    *ant = l->cabeca;  
    PONT atual = l->cabeca->prox;  
    l->cabeca->reg.chave = ch;  
    while (atual->reg.chave<ch) {  
        *ant = atual;  
        atual = atual->prox;  
    }  
    if (atual != l->cabeca && atual->reg.chave == ch) return atual;  
    return NULL;  
}
```

Inserção ordenada - sem duplicação

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {  
    PONT ant, i;  
    i = buscaSeqExc(l, reg.chave, &ant);  
    if (i != NULL) return false;  
    i = (PONT) malloc(sizeof(ELEMENTO));  
    i->reg = reg;  
    i->prox = ant->prox;  
    ant->prox = i;  
    return true;  
}
```

Exclusão de um elemento

- O usuário passa a chave do elemento que ele quer excluir
- Se houver um elemento com esta chave na lista, **exclui este elemento** da lista, **acerta os ponteiros** envolvidos e retorna *true*.
- Caso contrário, retorna *false*
- Para esta função precisamos saber quem é o **predecessor** do elemento a ser excluído.

Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {  
    PONT ant, i;  
    i = buscaSeqExc(l, ch, &ant);  
    if (i == NULL) return false;  
    ant->prox = i->prox;  
    free(i);  
    return true;  
}
```

Reinicialização da lista

Para reinicializar a estrutura, precisamos **excluir todos os elementos válidos** e atualizar o campo *prox* do nó cabeça.

```
void reinicializarLista(LISTA* l) {  
    PONT end = l->cabeca->prox;  
    while (end != l->cabeca) {  
        PONT apagar = end;  
        end = end->prox;  
        free(apagar);  
    }  
    l->cabeca->prox = l->cabeca;  
}
```


AULA 07

ESTRUTURA DE DADOS

Lista ligada circular com nó cabeça

Norton T. Roman & Luciano A. Digiampietri