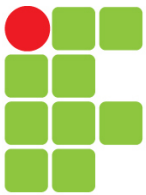
Listas duplamente encadeadas – *Doubly-linked List*

Programação de computadores II

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

25 de setembro de 2017



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópicos da aula

Tópico

Objetivos

- Entender o conceito de listas duplamente encadeadas – *Doubly-linked List*
- Aprender a utilizar esta estrutura de dados
- Aprender a implementar uma lista duplamente encadeada
- Utilizar o conceito de "dados abstratos e estruturados" para fornecer funções simples de manipulação

Tópico

Lista duplamente encadeada

Lista duplamente encadeada – *doubly-linked list* – é uma simples sequencia de dados dinamicamente alocados onde cada um aponta para o seu sucessor e antecessor.

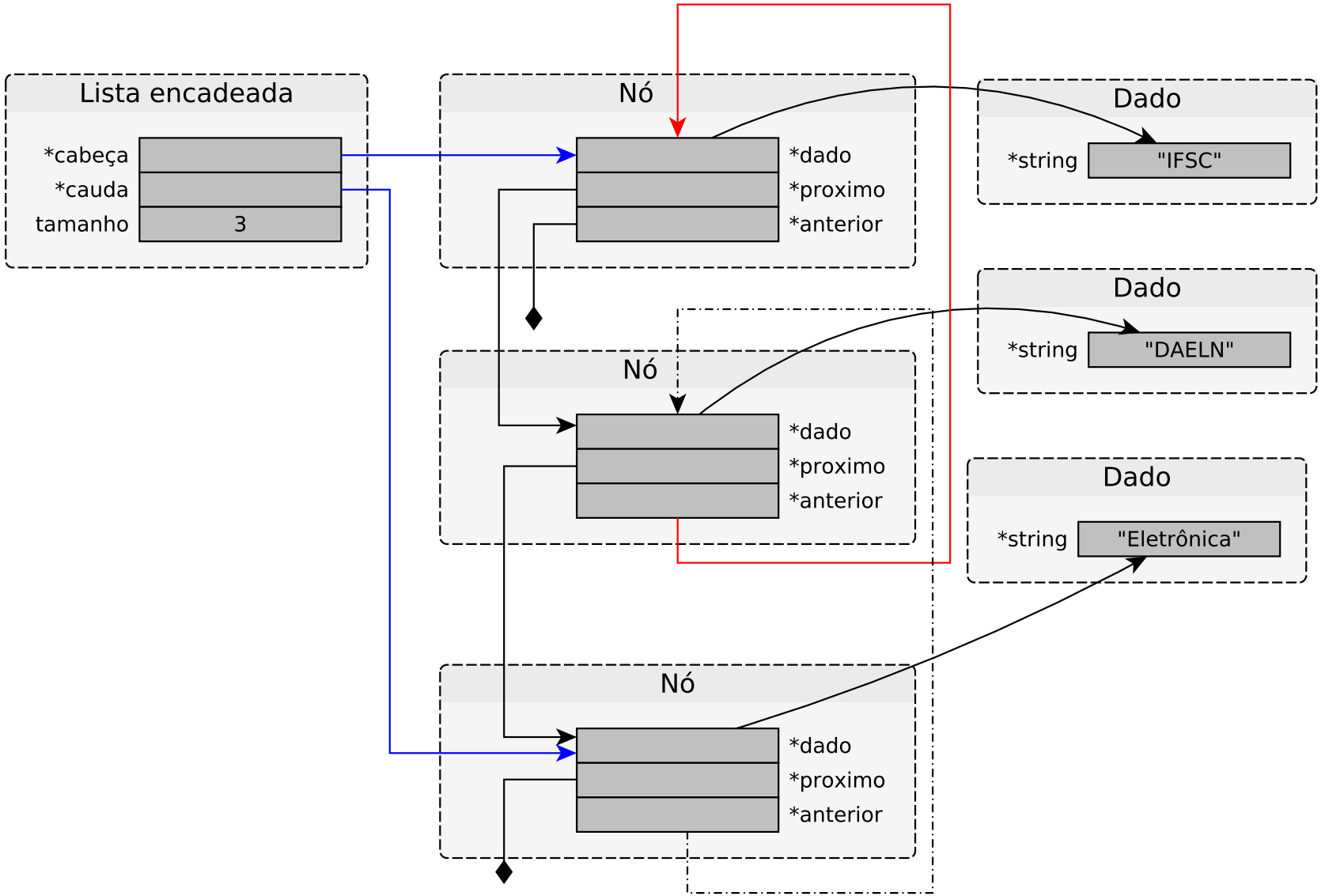
Funções básicas:

- *add_cabeca*: adiciona um elemento no inicio da lista
- *add_cauda*: adicionar uma elemento no final da lista
- *remove*: remove um elemento

Complexidade:

- *add_cabeca*: O(1)
- *add_cauda*: O(1)
- *remove*: O(1)

Lista duplamente encadeada

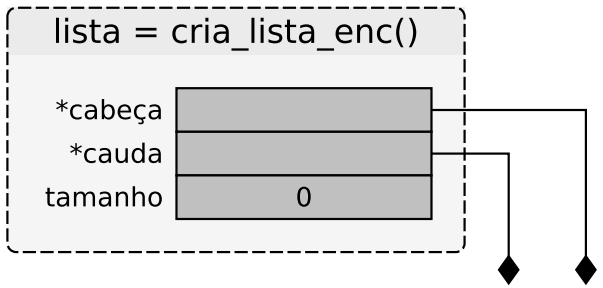


Tópico

Cria lista

```c
int main()
{

 lista_enc_t* lista = NULL;

 lista = cria_lista_enc();


}
```

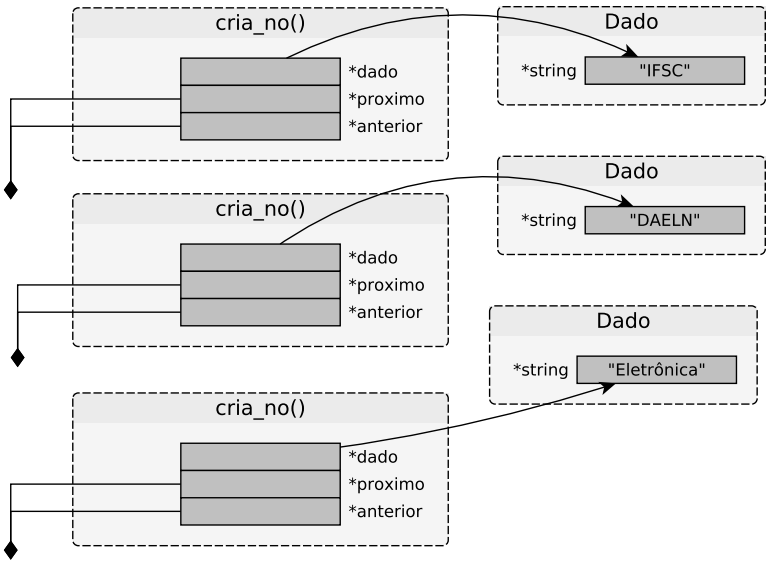
```c
lista_enc_t *cria_lista_enc (void) {
  lista_enc_t *p = malloc(sizeof(lista_enc_t));

  if (p == NULL){
      perror("cria_lista_enc:");
      exit(EXIT_FAILURE);
  }

  p->cabeca = NULL;
  p->cauda = NULL;
  p->tamanho = 0;

  return p;
}
```



Cria elementos

```c
int main()
{

 no_t* elemento[3];

 char nome_1[] = "IFSC";
 char nome_2[] = "DAELN";
 char nome_3[] = "Eletronica";

 elemento[0] = cria_no((void*)nome_1);
 elemento[1] = cria_no((void*)nome_2);
 elemento[2] = cria_no((void*)nome_3);

}
```

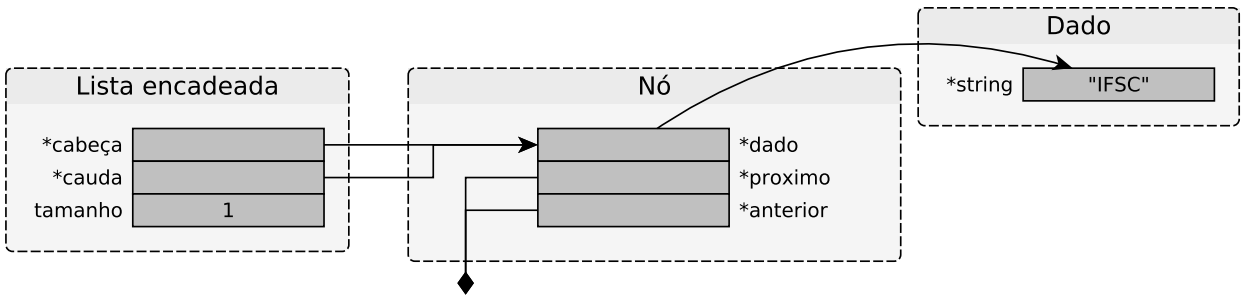
```c
no_t *cria_no(void *dado)
{
    no_t *p = malloc(sizeof(no_t));

    if (p == NULL){
        perror("cria_no:");
        exit(EXIT_FAILURE);
    }

    p->dados = dado;
    p->proximo = NULL;
    p->anterior = NULL;

    return p;
}
```



Adiciona na cauda

```c
int main()
{
  no_t* elemento = NULL;
  lista_enc_t* lista = NULL;

  char nome_1[] = "IFSC";
  char nome_2[] = "DAELN";
  char nome_3[] = "Eletronica";

  lista = cria_lista_enc();

  elemento = cria_no((void*)nome_1);
  add_cauda(lista, elemento);
}
```

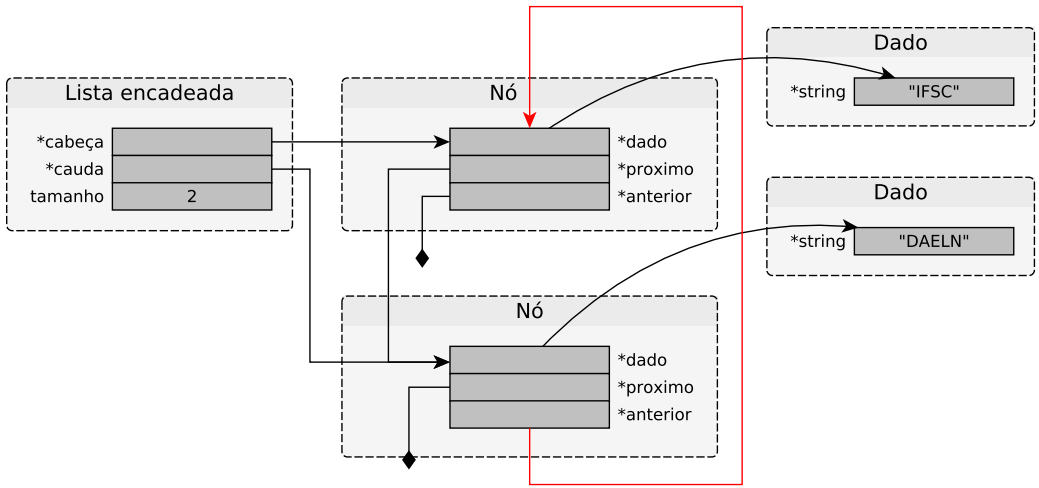
```c
void add_cauda(lista_enc_t *lista, no_t* elemento)
{
    //--- Controle de erros, null pointer aqui

    //lista vazia
    if (lista->tamanho == 0)   {
          lista->cauda = elemento;
          lista->cabeca = elemento;
          lista->tamanho++;
          desliga_no(elemento);   }
    else {
          desliga_no(elemento);
          liga_nos(lista->cauda, elemento);
          lista->cauda = elemento;
          lista->tamanho++;
    }
}
```



Adiciona na cauda

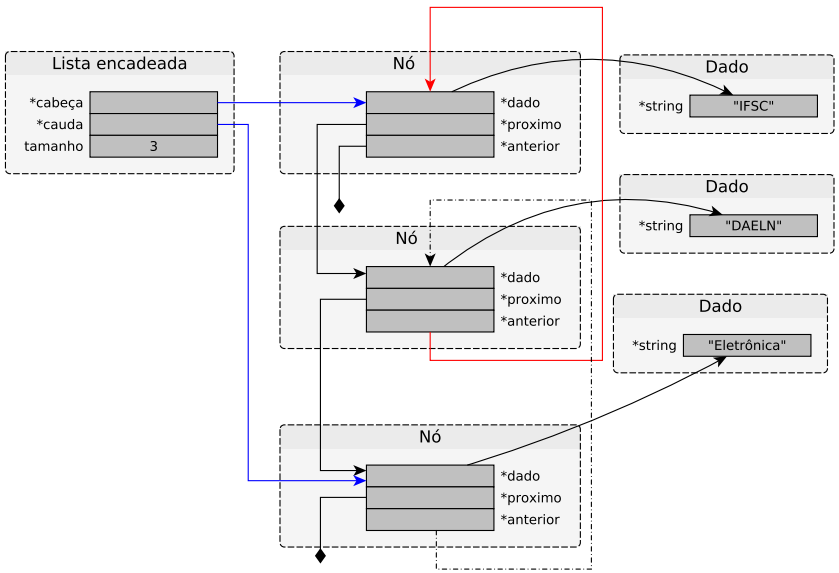
```c
int main()
{
  no_t* elemento = NULL;
  lista_enc_t* lista = NULL;

  char nome_1[] = "IFSC";
  char nome_2[] = "DAELN";
  char nome_3[] = "Eletronica";

  lista = cria_lista_enc();

  elemento = cria_no((void*)nome_1);
  add_cauda(lista, elemento);

  elemento = cria_no((void*)nome_2);
  add_cauda(lista, elemento);
}
```

```c
void add_cauda(lista_enc_t *lista, no_t* elemento)
{
   //--- Controle de erros, null pointer aqui

   //lista vazia
   if (lista->tamanho == 0)   {
        lista->cauda = elemento;
        lista->cabeca = elemento;
        lista->tamanho++;
        desliga_no(elemento);   }
   else {
        desliga_no(elemento);
        liga_nos(lista->cauda, elemento);
        lista->cauda = elemento;
        lista->tamanho++;
   }
}
```



Adiciona na cauda

```c
int main()
{
  no_t* elemento = NULL;
  lista_enc_t* lista = NULL;

  char nome_1[] = "IFSC";
  char nome_2[] = "DAELN";
  char nome_3[] = "Eletronica";

  lista = cria_lista_enc();

  elemento = cria_no((void*)nome_1);
  add_cauda(lista, elemento);

  elemento = cria_no((void*)nome_2);
  add_cauda(lista, elemento);

  elemento = cria_no((void*)nome_3);
  add_cauda(lista, elemento);
}
```



Tópico

Exemplo – no.c I

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

#include "no.h"

struct nos{
    void* dados;
    no_t *proximo;
    no_t *anterior;
};

// Cria um novo no
no_t *cria_no(void *dado)
{
    no_t *p = malloc(sizeof(no_t));

    if (p == NULL){
        perror("cria_no:");
        exit(EXIT_FAILURE);
    }

    p->dados = dado;
    p->proximo = NULL;
    p->anterior = NULL;

    return p;
}

void liga_nos (no_t *fonte, no_t *destino)
{
```

Exemplo – no.c II

```
    if (fonte == NULL || destino == NULL){
        fprintf(stderr,"liga_nos: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

    fonte->proximo = destino;
    destino->anterior = fonte;
}

void desliga_no (no_t *no)
{
    if (no == NULL) {
        fprintf(stderr,"liga_nos: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

    no->proximo = NULL;
    no->anterior = NULL;
}

void *obtem_dado (no_t *no)
{
    if (no == NULL) {
        fprintf(stderr,"liga_nos: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

    return no->dados;
}
```

Exemplo – no.c III

```
no_t *obtem_proximo (no_t *no)
{
    if (no == NULL) {
        fprintf(stderr,"liga_nos: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

    return no->proximo;
}

no_t *obtem_anterior (no_t *no)
{
    if (no == NULL) {
        fprintf(stderr,"liga_nos: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

    return no->anterior;
}
```

Exemplo – lista_enc.c I

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

#include "lista_enc.h"
#include "no.h"

#define DEBUG

struct listas_enc {
    no_t *cabeca;
    no_t *cauda;
    int tamanho;
};

//cria uma lista vazia
lista_enc_t *cria_lista_enc (void) {
    lista_enc_t *p = malloc(sizeof(lista_enc_t));

    if (p == NULL){
        perror("cria_lista_enc:");
        exit(EXIT_FAILURE);
    }

    p->cabeca = NULL;
    p->cauda = NULL;
    p->tamanho = 0;

    return p;
}
```

Exemplo – lista_enc.c II

```
void add_cauda(lista_enc_t *lista, no_t* elemento)
{
    if (lista == NULL || elemento == NULL){
        fprintf(stderr,"add_cauda: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

   #ifdef DEBUG
   printf("Adicionando %p --- tamanho: %d\n", elemento, lista->tamanho);
   #endif // DEBUG

   //lista vazia
   if (lista->tamanho == 0)
   {
        #ifdef DEBUG
        printf("add_cauda: add primeiro elemento: %p\n", elemento);
        #endif // DEBUG

        lista->cauda = elemento;
        lista->cabeca = elemento;
        lista->tamanho++;

        desliga_no(elemento);
   }
   else {
        // Remove qualquer ligacao antiga
        desliga_no(elemento);
        // Liga cauda da lista com novo elemento
        liga_nos(lista->cauda, elemento);
```

Exemplo – lista_enc.c III

```
        lista->cauda = elemento;
        lista->tamanho++;
    }
}

void imprimi_lista (lista_enc_t *lista)
{
    no_t *no = NULL;

    if (lista == NULL){
        fprintf(stderr,"imprimi_lista: ponteiros invalidos");
        exit(EXIT_FAILURE);
    }

    no = lista->cabeca;

    while (no){
        printf("Dados: %p\n", obtem_dado(no));

        no = obtem_proximo(no);
    }
}

void imprimi_lista_tras (lista_enc_t *lista)
{
    no_t *no = NULL;

    if (lista == NULL){
        fprintf(stderr,"imprimi_lista: ponteiros invalidos");
        exit(EXIT_FAILURE);
```

Exemplo – lista_enc.c IV

```
    }

    no = lista->cauda;

    while (no){
        printf("Dados: %p\n", obtem_dado(no));

        no = obtem_anterior(no);
    }
}
```

Tópico

Exercício

Baixe do Moodle a implementação exemplo da lista simplesmente encadeada e modifique para duplamente encadeada. Sua lista deve suportar:

- *add_cabeca*: adiciona um elemento no inicio da lista
- *add_cauda*: adiciona um elemento no inicio da lista
- *remove_cabeca*: remove um elemento
- *remove_cauda*: remove um elemento
- *remove_pos*: remove um elemento em uma posição arbitrária da lista
- *tamanho*: retorna tamanho da lista
- *vazia*: retorna se lista está vazia

Exercício

Implemente uma lista duplamente encadeada utilizando apenas um ponteiro por nó ao invés de implementação padrão (*proximo* e *anterior*). Dicas:

- Valores dos ponteiros podem ser interpretados como inteiros de *k bits*.
- no.xprox = no.proximo XOR no.anterior.
- null = 0.

Certifique-se de descrever qual informação você necessita para acessar a cabeça da lista. Implemente também os procedimentos *buscar*, *inserir* e *remover* nesta nova lista. Mostre também como navegar na lista de forma reversa.