Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

AULA 13

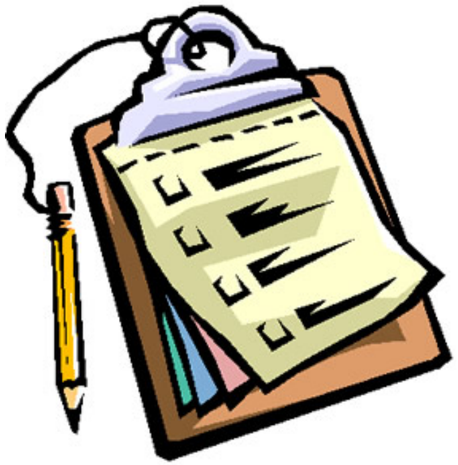
Estrutura de dados básico I (EDB1)

Prof. Msc. Janiheryson Felipe (Felipe)

Natal, RN
2023

OBJETIVOS DA AULA

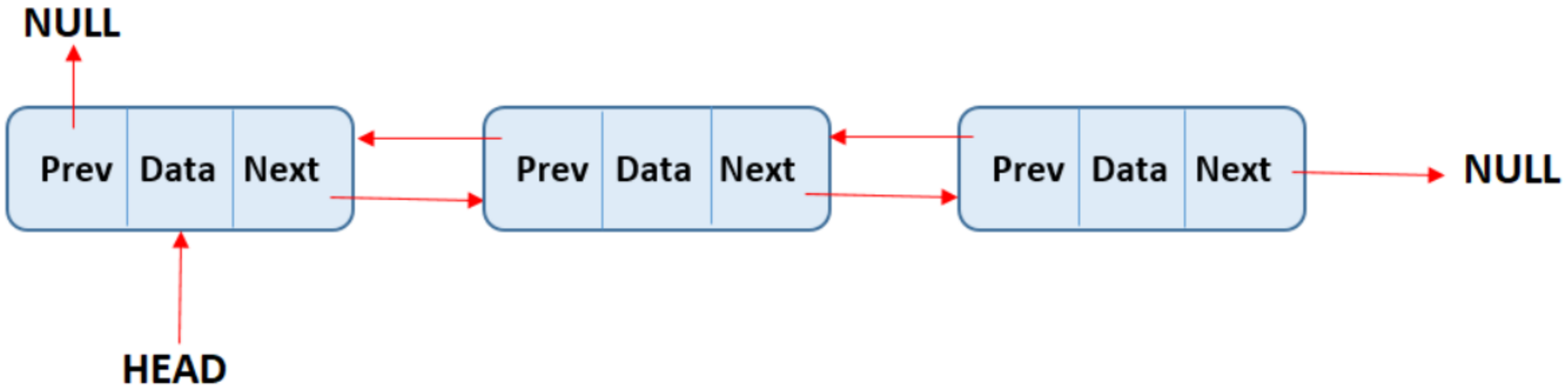
- Apresentar os conceitos de lista duplamente encadeada e sua implementação em memória.
  - Conhecer as listas duplamente encadeada;
  - Implementar uma lista duplamente encadeada a partir do zero;



LISTAS DUPLAMENTE LIGADA

LISTAS DUPLAMENTE LIGADAS (DOUBLE LINKED LIST)

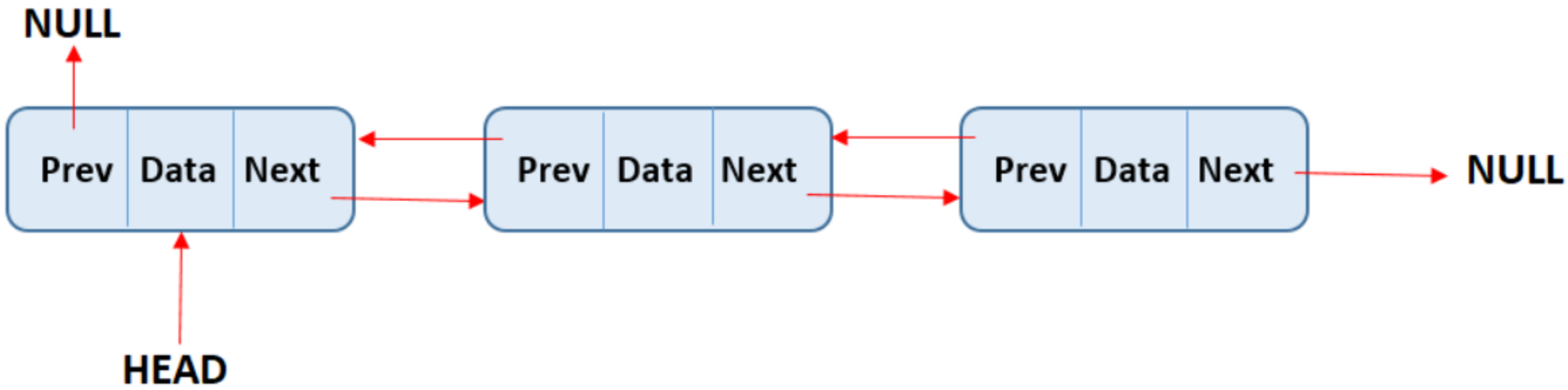
Uma lista encadeada dupla é uma lista bidirecional na qual todos os nós terão dois links. Isso ajuda a acessar o nó sucessor e o nó predecessor a partir da posição do nó fornecida. Ele fornece deslocamento bidirecional.



LISTAS DUPLAMENTE LIGADAS (DOUBLE LINKED LIST)

Cada nó contém três campos:

- ponteiro esquerdo;
- valor;
- ponteiro direito;



LISTAS DUPLAMENTE LIGADAS (DOUBLE LINKED LIST)

O ponteiro esquerdo aponta para o nó antecessor e o ponteiro direito aponta para o nó sucessor. O campo de valor armazena os dados necessários.

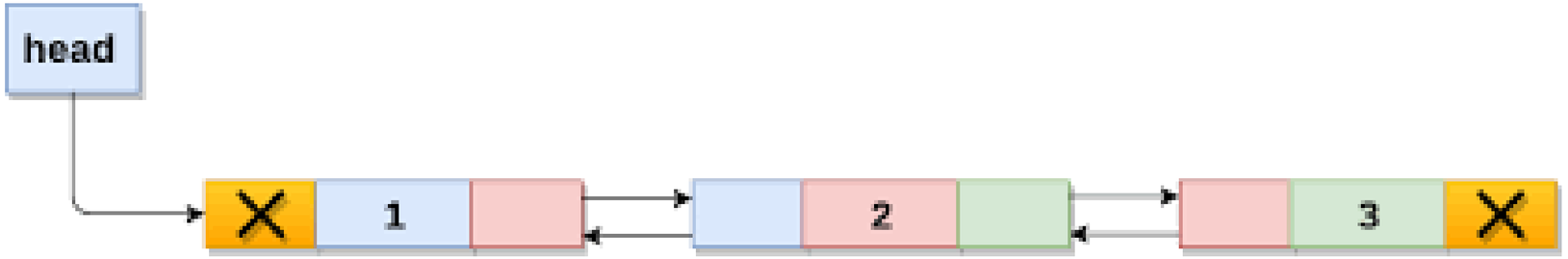
Muitos aplicativos requerem pesquisa para frente e para trás nos nós de uma lista.

- Por exemplo, procurar um nome em uma lista telefônica exigiria uma varredura para frente e para trás em uma região de toda a lista.

LISTAS DUPLAMENTE LIGADAS (DOUBLE LINKED LIST)

As operações básicas em uma lista encadeada dupla são:

- Criação da lista.
- Inserção de nós na lista
- Eliminação de nós.
- impressão da lista.



Doubly Linked List

CRIAÇÃO DE UMA LDL

A criação de uma lista encadeada dupla começa com a criação de um nó.

- Memória suficiente deve ser alocada para criar um nó.
- As informações são armazenadas na memória, alocadas usando a função malloc().
- A função push() é usada para criar um nó, depois de alocar memória para a estrutura do tipo nó, as informações do item (ou seja, dados) devem ser lidas do usuário e definir o campo esquerdo como NULL e o campo direito também definido como NULL.

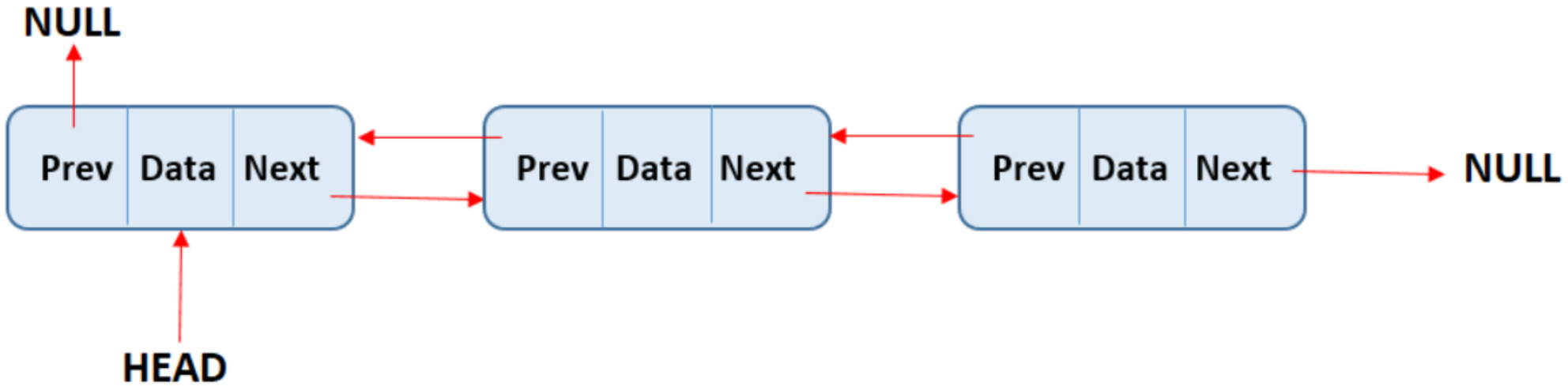
CRIAÇÃO DE UMA LDL

```
typedef struct no {
    int valor;
    no* prev;
    no* next;
}No;
```

INSERÇÃO DE VALORES

Podemos inserir valores em uma lista duplamente encadeada basicamente de tres formas possiveis:

- No inicio da lista
- No fim da lista
- Em uma posição definida



```c
void push_front(No **list, int x){
  //Cria um novo nó
  No *novoNo = (No*)malloc(sizeof(No));
  novoNo->valor = x;
  novoNo->next = NULL;
  novoNo->prev = NULL;
  //Verifica se a lista ta vazia
  if(*list == NULL){
    *list = novoNo;
    return;
  }
  //Se a lista não for vazia
  novoNo->next = *list;
  *list = novoNo;
}
```

INSERÇÃO DE UM NÓ NA LISTA

NA FRENTE

```c
void push_back(No **list, int x){
  //Cria um novo nó
  No *novoNo = (No*)malloc(sizeof(No));
  novoNo->valor = x;
  novoNo->next = NULL;
  novoNo->prev = NULL;

  //Verifica se a lista ta vazia
  if(*list == NULL){
    *list = novoNo;
    return;
  }
  //Se a lista não for vazia
  No* temp = *list;
  while(temp->next != NULL){
    temp = temp->next;
  }
  temp->next = novoNo;
  novoNo->prev = temp;
}
```

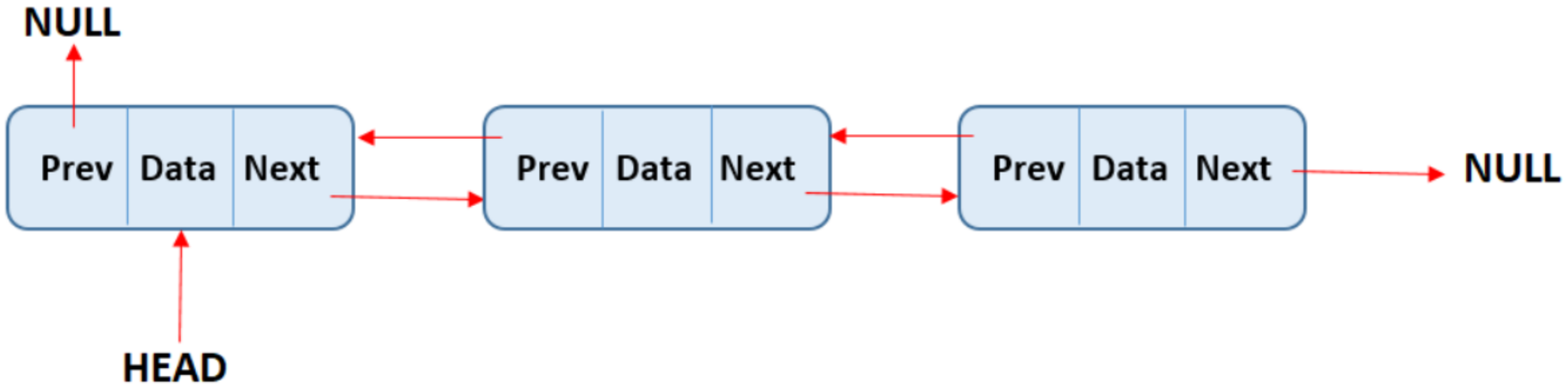
INSERÇÃO DE UM NÓ NA LISTA

NO FIM

EXCLUSÃO DE NÓS

Podemos excluir valores em uma lista duplamente encadeada basicamente de tres formas possiveis:

- No inicio da lista
- No fim da lista
- Em uma posição definida ou um valor qualquer (ideal)



```
void pop_front(No **list){
  if(*list == NULL){
    printf("A lista está vazia\n");
    return;
  }
  if(size(*list) == 1){
    free(*list);
    *list = nullptr;
    return;
  }
  No *temp = *list;
  temp->prev = NULL;
  *list = temp->next;
  free(temp);
}
```

EXCLUSÃO DE UM NÓ NA LISTA

NO INÍCIO

```
void pop_back(No **list){
  if(*list == NULL){
    printf("A lista está vazia\n");
    return;
  }
  if(size(*list) == 1){
    free(*list);
    *list = nullptr;
    return;
  }
  No* temp = *list;
  while(temp->next != NULL){
    temp = temp->next;
  }
  temp->prev->next = NULL;
  free(temp);
}
```

EXCLUSÃO DE UM NÓ NA LISTA

NO FIM

PROCURAR UM VALOR NA LISTA

```
list* find(int x, list *p){
    if(p == NULL){
      return NULL;
    }
   if(p->valor == x){
     return p;
   }else{
     return find(x, p->prox);
   }
}
```

IMPRIMIR OS VALORES DA LISTA

```
void print(list *p){
   if(p == NULL){
      return;
   }else{
         printf("%d\n", p->valor);
         print(p->prox);
   }
}
```

TAMANHO DA LISTA

```
int size(list *p){
   if(p->prox == NULL){
      return 1;
   }else{
     return size(p->prox) + 1;
   }
}
```

DÚVIDAS???