Listas Encadeadas

Simplesmente Duplamente Circulares

TAD – Tipo Abstrato de Dados

Tipo Abstrato de Dados (TAD) é um tipo (ou classe) para objetos cujo comportamento é definido por um conjunto de valores e um conjunto de operações. A definição de TAD apenas menciona quais operações devem ser executadas, mas não como essas operações serão implementadas.

Não especifica como os dados serão organizados na memória e quais algoritmos serão usados para implementar as operações.

Dá uma visão independente de implementação.

O processo de fornecer apenas o essencial e esconder os detalhes é conhecido como abstração.

TAD – Tipo Abstrato de Dados

Podemos pensar no TAD como uma caixa preta que esconde a estrutura interna e o design do tipo de dados.

TAD – Lista Encadeada

Qual é a limitação de um array?

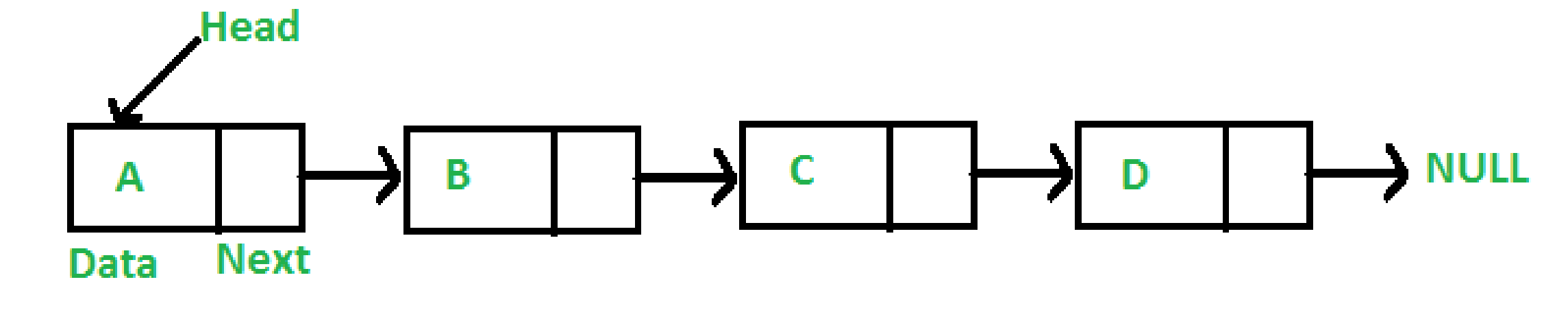
- Tamanho Fixo
- Insert e Delete em uma determinada posição, é complexo.

TAD – Lista Encadeada

Vantagens da Lista Encadeada?

- Tamanho não é fixo
- Insert e Delete simples e fácil

Lista Simplesmente Encadeada



Lista Simplesmente Encadeada

Vantagens da Lista Encadeada(sobre o array)

- Tamanho Dinâmico
- Facilidade no Insert e Delete

Lista Simplesmente Encadeada

Desvantagens:

- Acesso aleatório não permitido
- É necessário espaço de memória extra para um ponteiro com cada elemento da lista

Lista Simplesmente Encadeada

Representação no C:

- Uma lista simpl. encadeada é representada por uma estrutura chamada de node, onde há um ponteiro que ‘aponta’ para o próximo node da lista.
- O primeiro nó(ou node) é chamado de HEAD.
- Se a lista está vazia, então o valor de HEAD é NULL.

Lista Simplesmente Encadeada

Representação no C:

Cada node da lista consiste, em pelo menos duas partes:

1) O Dado armazenado
2) Um ponteiro para o próximo node.

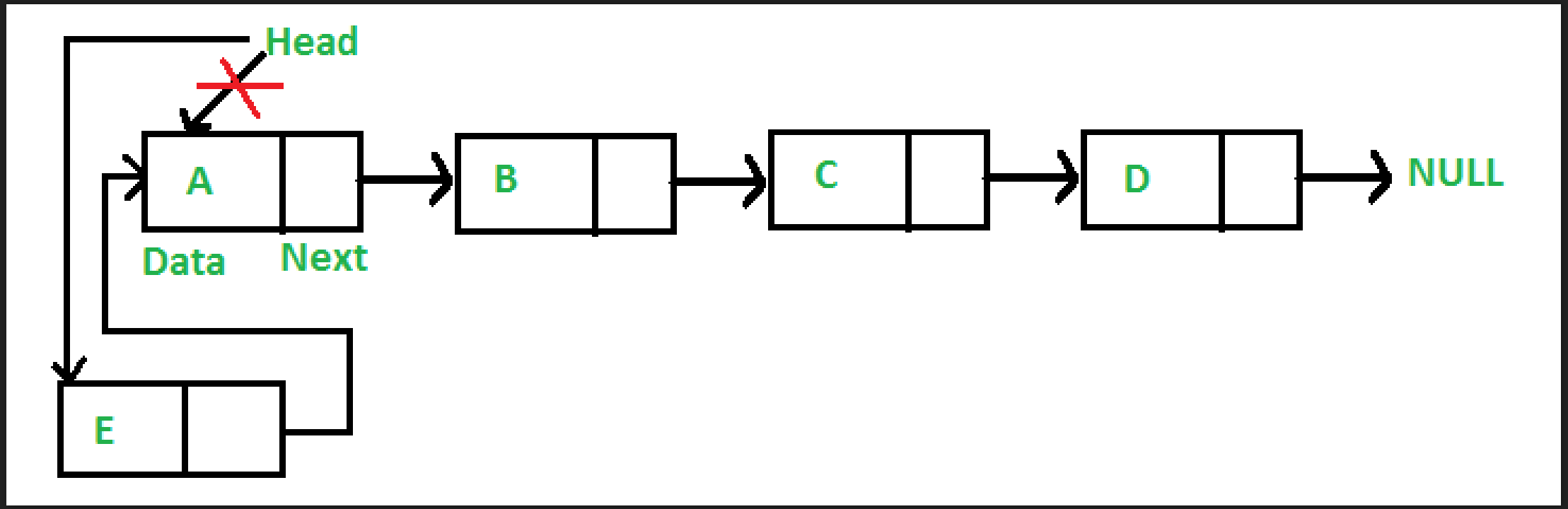
Exemplo:

```c
#include<stdlib.h>

int main(){

   struct Node{
      int data;
      struct Node *next;
   };

   return 1;
}
```

Outro Exemplo

goto codeblocks

Adicionando no início da lista



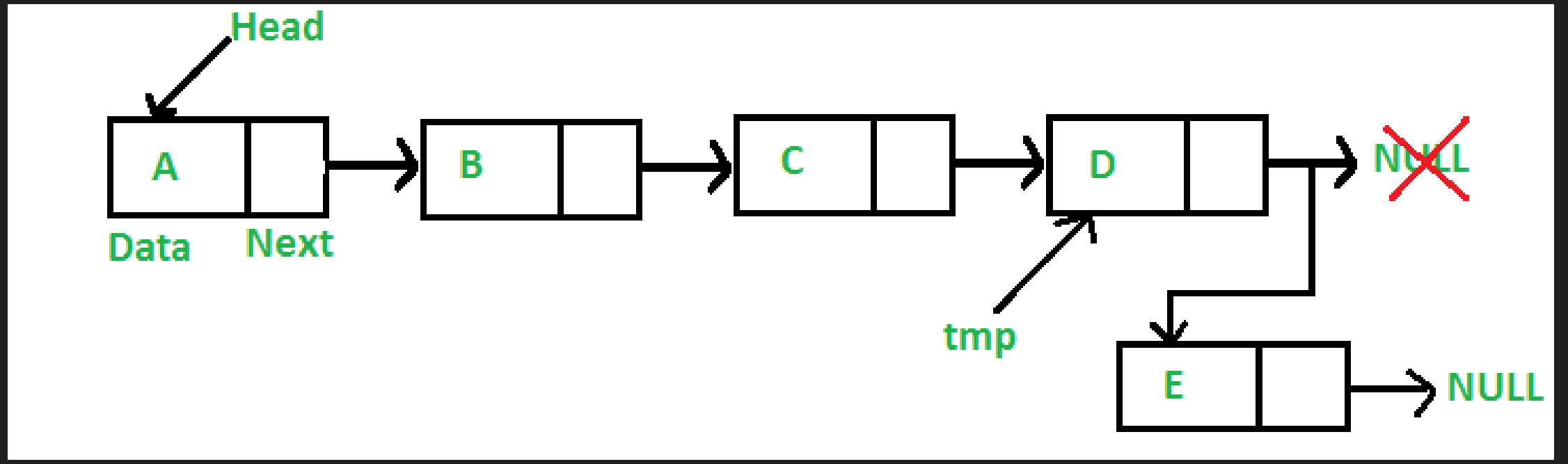
Passo a passo

- Aloca o node(malloc)
- Adiciona o novo dado
- Faça o novo node como HEAD
- Mova o ponteiro do HEAD para o novo node

Complexidade

- Custo para adicionar no início da lista: O(1)

Adicionando no fim da lista



Passo a passo

- Aloca o node(malloc)
- Adiciona o novo dado
- Aponte o novo node para NULL
- Se lista está vazia, então faça o novo node como HEAD
- Senão percorra a lista até o último node
- Aponte o último para o novo node.

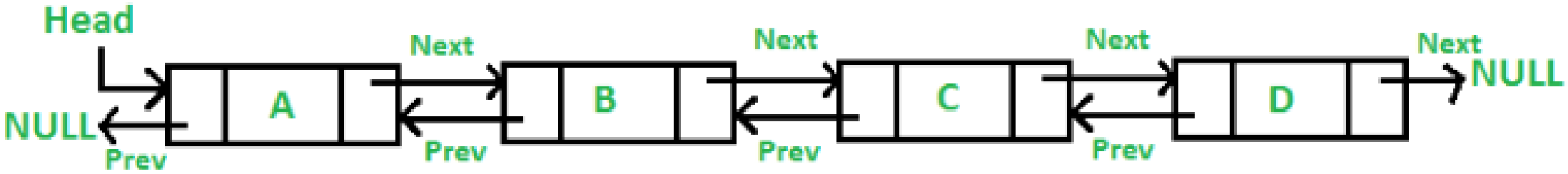
Complexidade

- Custo para adicionar no final da lista: O(n)

Lista Duplamente encadeada

A lista duplamente encadeada traz em sua estrutura um ponteiro extra, chamado de ‘anterior’

Lista Duplamente encadeada



Lista Duplamente encadeada

```
/* Definindo a estrutura de dados da lista duplamente encadeada  */
struct Node
{
  int data;
  struct Node *anterior; // aponta para o 'node' anterior
  struct Node *proximo; // aponta para o próximo 'node' da lista
};
```

Lista Duplamente encadeada

Vantagens sobre a Lista Simpl. Encadeda:

- Pode ser percorrida nos dois sentidos
- A operação de delete é mais eficiente

Lista Duplamente encadeada

Desvantagens sobre a Lista Simpl. Encadeda:

- Espaço extra para outro ponteiro
- Todas as operações(insert, delete) tem mais um dado a ser mantido.

Exercício

Faça as operações de inserir no início, deletar no início e remover no fim