Estrutura de dados lista encadeada em C

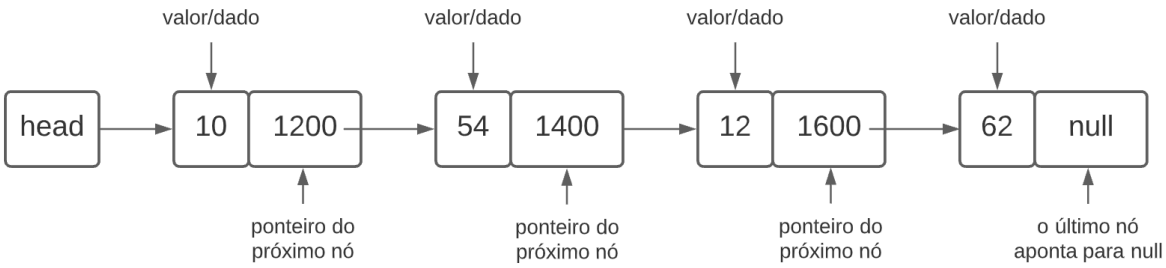
Oque e um nó?

Um **nó** é um dos elementos básicos que compõem uma **lista encadeada** (ou outras estruturas de dados, como árvores). Em uma lista encadeada, cada nó contém dois componentes principais:

1. **Dados**: A informação que o nó armazena. Por exemplo, em uma lista encadeada de números inteiros, o nó pode armazenar um valor como `3`, `5` ou `10`.
2. **Ponteiro (ou referência)**: Um ponteiro que aponta para o **próximo nó** da lista. Isso cria o encadeamento entre os nós. Se o nó for o último da lista, esse ponteiro geralmente tem o valor `NULL`, indicando o fim da lista.

Aqui está uma representação visual simples de um nó em uma lista encadeada:

```
[ Dado | Ponteiro ] -> [ Dado | Ponteiro ] -> [ Dado | NULL ]
```



Cada bloco representa um nó, onde o primeiro campo é o dado armazenado e o segundo campo é o ponteiro para o próximo nó.

Exemplo de um Nó em C

Em C, um nó pode ser definido usando uma **estrutura** (`struct`). Por exemplo, em uma lista encadeada de inteiros:

```
struct Node {
    int data;           // O dado armazenado no nó
    struct Node* next;  // Ponteiro para o próximo nó
};
```

O Papel do Nó em uma Lista Encadeada

- **Cabeça (head)**: O primeiro nó de uma lista encadeada. Para acessar a lista, você começa por ele.
- **Último Nó**: O último nó tem seu ponteiro apontando para `NULL`, indicando que não há mais nós após ele.

Exemplo Visual

Imagine que temos uma lista encadeada com três nós contendo os valores 3, 1 e 5:

```
[ 3 | * ] -> [ 1 | * ] -> [ 5 | NULL ]
```

Aqui:

- O primeiro nó armazena o valor 3 e aponta para o segundo nó.
- O segundo nó armazena o valor 1 e aponta para o terceiro nó.
- O terceiro nó armazena o valor 5 e seu ponteiro aponta para `NULL`, indicando o fim da lista.

Importância dos Nós

Os nós permitem que listas encadeadas sejam **dinâmicas**. Diferente de arrays, onde o tamanho é fixo, em listas encadeadas você pode adicionar ou remover nós conforme necessário, sem precisar definir um tamanho fixo antecipadamente.

1. Estrutura de um nó da lista encadeada

A lista encadeada é composta de nós, onde cada nó armazena dois elementos:

- **Dado**: A informação (por exemplo, um número inteiro).
- **Ponteiro**: Um ponteiro que aponta para o próximo nó da lista.

Aqui está como você pode definir essa estrutura em C:

```
// Definindo a estrutura do nó
struct Node {
    int data;           // Dado armazenado no nó
    struct Node* next;  // Ponteiro para o próximo nó
};
```

2. Inserindo um nó no início da lista

Agora, vamos implementar a primeira função que **insere um nó no início da lista**. Vamos usar a alocação dinâmica de memória com `malloc` para criar um novo nó.

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

// Estrutura do nó
struct Node {
    int data;
    struct Node* next;
};

// Função para adicionar um nó no início da lista
void insertAtBeginning(struct Node** head, int newData) {
    // Aloca memória para o novo nó
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));

    // Coloca o dado no novo nó
    newNode->data = newData;

    // Faz o novo nó apontar para o antigo primeiro nó
    newNode->next = *head;

    // Atualiza o ponteiro da cabeça para o novo nó
    *head = newNode;
}
```

3. Imprimindo a lista encadeada

Agora, precisamos de uma função para percorrer a lista e imprimir seus elementos. Vamos implementar a função `printList`.

```
// Função para percorrer e imprimir a lista
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d -> ", node->data); // Imprime o dado do nó atual
        node = node->next;            // Vai para o próximo nó
    }
    printf("NULL\n"); // Indica o final da lista
```

```
}
```

4. Testando a inserção e impressão

Agora vamos juntar tudo no programa principal e testar a inserção de novos nós na lista e a impressão da lista:

```
int main() {
    struct Node* head = NULL; // Inicialmente, a lista está vazia

    // Inserindo elementos no início da lista
    insertAtBeginning(&head, 1);  // Lista: 1 -> NULL
    insertAtBeginning(&head, 2);  // Lista: 2 -> 1 -> NULL
    insertAtBeginning(&head, 3);  // Lista: 3 -> 2 -> 1 -> NULL

    // Imprimindo a lista encadeada
    printList(head); // Saída esperada: 3 -> 2 -> 1 -> NULL

    return 0;
}
```

5. Inserindo um nó no final da lista

Agora, vamos criar uma função para inserir um novo nó no **final da lista**:

```
// Função para adicionar um nó no final da lista
void insertAtEnd(struct Node** head, int newData) {
   // Aloca memória para o novo nó
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    struct Node* last = *head; // Cria um ponteiro temporário para
percorrer a lista

    newNode->data = newData;   // Coloca o dado no novo nó
    newNode->next = NULL;      // O novo nó será o último, então seu
ponteiro será NULL

    // Se a lista estiver vazia, o novo nó será o primeiro
    if (*head == NULL) {
        *head = newNode;
        return;
```

```
    }

    // Percorre até o último nó
    while (last->next != NULL) {
        last = last->next;
    }

    // Faz o último nó apontar para o novo nó
    last->next = newNode;
}
```

6. Removendo um nó da lista

Outra operação importante é a remoção de um nó. Vamos criar uma função para remover o primeiro nó que contém um determinado valor.

```
// Função para remover o primeiro nó que contém o valor 'key'
void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    struct Node* prev = NULL;

    // Se o nó a ser removido for o primeiro
    if (temp != NULL && temp->data == key) {
        *head = temp->next; // Muda o ponteiro da cabeça para o
próximo nó
        free(temp);         // Libera a memória do nó
        return;
    }

    // Procura o nó com o valor 'key'
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // Se o valor não foi encontrado
    if (temp == NULL) return;

    // Desvincula o nó da lista e libera sua memória
    prev->next = temp->next;
    free(temp);
```

```
}
```

7. Testando as novas operações

] Agora, você pode testar a inserção no final e a remoção de nós:

```
int main() {
    struct Node* head = NULL;

    // Inserindo elementos
    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);

    // Imprime a lista encadeada
    printf("Lista após inserção: ");
    printList(head); // Esperado: 1 -> 2 -> 3 -> NULL

    // Remove o nó com valor 2
    deleteNode(&head, 2);

    // Imprime a lista após a remoção
    printf("Lista após remoção do 2: ");
    printList(head); // Esperado: 1 -> 3 -> NULL

    return 0;
}
```