Entendendo Listas Encadeadas em C Puro

O que é uma Lista Encadeada?

Uma lista encadeada é uma estrutura de dados linear, assim como um array, mas que não armazena seus elementos em locais de memória contíguos. Em vez disso, cada elemento (chamado de **nó**) contém não apenas os dados em si, mas também um ponteiro (ou referência) para o próximo nó na sequência. Essa abordagem permite uma alocação de memória dinâmica e flexível, onde os nós podem ser espalhados pela memória e conectados por meio desses ponteiros.

Comparação com Arrays:

Característica	Lista Encadeada	Array
Alocação de Memória	Dinâmica (nós alocados individualmente)	Estática ou Dinâmica (bloco contíguo)
Acesso aos Elementos	Sequencial (do início ao fim)	Direto (por índice)
Inserção/Remoção	Eficiente (altera apenas ponteiros)	Ineficiente (requer deslocamento de elementos)
Tamanho	Flexível (cresce e diminui conforme necessário)	Fixo (definido na criação)
Uso de Memória	Maior (ponteiros adicionais)	Menor (apenas dados)

Tipos de Listas Encadeadas

Existem diferentes variações de listas encadeadas, cada uma com suas particularidades:

1. Lista Simplesmente Encadeada (Singly Linked List)

É o tipo mais básico. Cada nó aponta apenas para o próximo nó na sequência. O último nó aponta para `NULL`, indicando o fim da lista. A navegação é possível apenas em uma direção (para frente).

```c
struct Node {
    int data;          // Dados armazenados no nó
    struct Node *next; // Ponteiro para o próximo nó
};
```

2. Lista Duplamente Encadeada (Doubly Linked List)

Cada nó possui dois ponteiros: um para o próximo nó (`next`) e outro para o nó anterior (`prev`). Isso permite a navegação em ambas as direções (para frente e para trás). A inserção e remoção podem ser ligeiramente mais complexas devido à necessidade de gerenciar dois ponteiros por nó, mas oferecem maior flexibilidade.

```c
struct Node {
    int data;
    struct Node *next; // Ponteiro para o próximo nó
    struct Node *prev; // Ponteiro para o nó anterior
};
```

3. Lista Encadeada Circular (Circular Linked List)

Neste tipo, o último nó da lista aponta para o primeiro nó, formando um ciclo. Pode ser simplesmente ou duplamente encadeada. A ausência de um `NULL` no final da lista requer um tratamento especial para evitar loops infinitos ao percorrer a lista.

A Estrutura de um Nó

Como mencionado, o elemento fundamental de uma lista encadeada é o **nó**. Cada nó é uma estrutura que geralmente contém:

- **Dados (data):** O valor real que a lista está armazenando. Pode ser de qualquer tipo de dado (inteiro, caractere, estrutura, etc.).
- **Ponteiro para o próximo (next):** Um ponteiro para o próximo nó na sequência. Em C, isso é tipicamente um ponteiro para a própria estrutura do nó (`struct`

`Node *next`).

Vantagens e Desvantagens das Listas Encadeadas

Vantagens:

- **Alocação de memória dinâmica:** A lista pode crescer ou diminuir em tempo de execução, conforme a necessidade, sem a necessidade de realocar um bloco contíguo de memória.
- **Inserção e remoção eficientes:** Adicionar ou remover elementos no meio da lista é muito eficiente, pois requer apenas a alteração de alguns ponteiros, sem a necessidade de deslocar outros elementos (como em arrays).
- **Uso eficiente da memória:** A memória é alocada apenas para os nós que realmente contêm dados, evitando desperdício de espaço.

Desvantagens:

- **Acesso sequencial:** Para acessar um elemento específico, é necessário percorrer a lista desde o início até encontrar o nó desejado. Isso torna o acesso a elementos menos eficiente do que em arrays (onde o acesso é direto por índice).
- **Maior uso de memória:** Cada nó requer memória adicional para armazenar o(s) ponteiro(s), o que pode ser um fator em sistemas com memória muito limitada.
- **Complexidade de implementação:** A manipulação de ponteiros pode ser mais complexa e propensa a erros (como vazamentos de memória ou ponteiros nulos) do que a manipulação de arrays.

Implementação da Lista Encadeada em C

Agora, vamos ver a implementação prática de uma lista simplesmente encadeada em C, incluindo as estruturas e as funções para as operações básicas.

Arquivo `lista_encadeada.h`

Este arquivo de cabeçalho define as estruturas `Node` e `LinkedList`, além de declarar os protótipos das funções que manipulam a lista.

```c
#ifndef LISTA_ENCADEADA_H
#define LISTA_ENCADEADA_H

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

// Definição da estrutura do nó
typedef struct Node {
    int data;          // Dados armazenados no nó
    struct Node *next; // Ponteiro para o próximo nó
} Node;

// Definição da estrutura da lista (cabeça)
typedef struct LinkedList {
    Node *head;        // Ponteiro para o primeiro nó da lista
    int size;          // Tamanho atual da lista
} LinkedList;

// Protótipos das funções
Node* createNode(int data);
void initList(LinkedList* list);
void insertAtBeginning(LinkedList* list, int data);
void insertAtEnd(LinkedList* list, int data);
void removeFromBeginning(LinkedList* list);
void removeFromEnd(LinkedList* list);
Node* searchElement(LinkedList* list, int data);
void printList(LinkedList* list);
void freeList(LinkedList* list);

#endif // LISTA_ENCadeada_H
```

Arquivo `lista_encadeada.c`

Este arquivo contém a implementação das funções declaradas no cabeçalho, que realizam as operações de criação de nó, inicialização, inserção, remoção, busca, impressão e liberação da lista.

```c
#include "lista_encadeada.h"

// Função para criar um novo nó
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        perror("Erro ao alocar memória para o nó");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Função para inicializar a lista
void initList(LinkedList* list) {
    list->head = NULL;
    list->size = 0;
}

// Função para inserir um nó no início da lista
void insertAtBeginning(LinkedList* list, int data) {
    Node* newNode = createNode(data);
    newNode->next = list->head;
    list->head = newNode;
    list->size++;
    printf("Elemento %d inserido no início.\n", data);
}

// Função para inserir um nó no final da lista
void insertAtEnd(LinkedList* list, int data) {
    Node* newNode = createNode(data);
    if (list->head == NULL) {
        list->head = newNode;
    } else {
        Node* current = list->head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
    list->size++;
    printf("Elemento %d inserido no final.\n", data);
}

// Função para remover um nó do início da lista
void removeFromBeginning(LinkedList* list) {
    if (list->head == NULL) {
        printf("A lista está vazia. Nada para remover.\n");
        return;
    }
    Node* temp = list->head;
    list->head = list->head->next;
    printf("Elemento %d removido do início.\n", temp->data);
    free(temp);
    list->size--;
}

// Função para remover um nó do final da lista
void removeFromEnd(LinkedList* list) {
    if (list->head == NULL) {
```

```
        printf("A lista está vazia. Nada para remover.\n");
        return;
    }
    if (list->head->next == NULL) { // Apenas um nó na lista
        printf("Elemento %d removido do final.\n", list->head->data);
        free(list->head);
        list->head = NULL;
        list->size--;
        return;
    }
    Node* current = list->head;
    while (current->next->next != NULL) {
        current = current->next;
    }
    printf("Elemento %d removido do final.\n", current->next->data);
    free(current->next);
    current->next = NULL;
    list->size--;
}

// Função para buscar um elemento na lista
Node* searchElement(LinkedList* list, int data) {
    Node* current = list->head;
    while (current != NULL) {
        if (current->data == data) {
            return current;
        }
        current = current->next;
    }
    return NULL; // Elemento não encontrado
}

// Função para imprimir a lista
void printList(LinkedList* list) {
    if (list->head == NULL) {
        printf("A lista está vazia.\n");
        return;
    }
    Node* current = list->head;
    printf("Elementos da lista (%d): ", list->size);
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

// Função para liberar a memória da lista
void freeList(LinkedList* list) {
    Node* current = list->head;
    Node* nextNode;
    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
    list->head = NULL;
    list->size = 0;
    printf("Memória da lista liberada.\n");
}
```

Arquivo `main.c`

Este é o programa principal que demonstra o uso das funções da lista encadeada.

```c
#include "lista_encadeada.h"

int main() {
    LinkedList myList;
    initList(&myList);

    // Inserir elementos no início
    insertAtBeginning(&myList, 10);
    insertAtBeginning(&myList, 20);
    insertAtBeginning(&myList, 30);
    printList(&myList);

    // Inserir elementos no final
    insertAtEnd(&myList, 5);
    insertAtEnd(&myList, 15);
    printList(&myList);

    // Buscar um elemento
    Node* found = searchElement(&myList, 20);
    if (found != NULL) {
        printf("Elemento 20 encontrado na lista.\n");
    } else {
        printf("Elemento 20 não encontrado na lista.\n");
    }

    found = searchElement(&myList, 99);
    if (found != NULL) {
        printf("Elemento 99 não encontrado na lista.\n");
    }

    // Remover do início
    removeFromBeginning(&myList);
    printList(&myList);

    // Remover do final
    removeFromEnd(&myList);
    printList(&myList);

    // Liberar a memória da lista
    freeList(&myList);
    printList(&myList); // Deve mostrar que a lista está vazia

    return 0;
}
```