Lista Duplamente Ligada

Introdução

Uma lista duplamente ligada é uma estrutura de dados linear que consiste em uma sequência de elementos, chamados de `Node`, onde cada nó armazena um valor e duas referências, uma para o próximo nó da sequência e outra para o nó anterior da sequência. A lista duplamente ligada é uma estrutura de dados dinâmica, ou seja, ela pode crescer ou diminuir de tamanho durante a execução do programa.

Estrutura

A lista duplamente ligada é composta por nós, onde cada nó contém um valor e duas referências, uma para o próximo nó e outra para o nó anterior.

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

struct Node {
    int data;
    struct Node* proximo;
    struct Node* anterior;
};
```

```
struct Node* inicio_da_lista = NULL;

void criar_lista()
{
    inicio_da_lista = NULL;
}
```

Inserção

A inserção de um novo nó em uma lista duplamente ligada pode ser feita no início, no final ou em qualquer posição da lista.

Inserção no início

Para inserir um novo nó no início da lista, basta criar um novo nó, atribuir o valor desejado a ele e ajustar as referências do novo nó e do nó que era o primeiro da lista.

```
void adicionar_elemento(int valor)
{
    struct Node *novo_node = (struct Node *)malloc(sizeof(struct Node));
    novo_node->data = valor;
    novo_node->proximo = inicio_da_lista;
    novo_node->anterior = NULL;
    if (inicio_da_lista != NULL)
    {
        inicio_da_lista->anterior = novo_node;
    }
    inicio_da_lista = novo_node;
}
```

Perceba que, ao adicionar um novo nó no início da lista, é necessário verificar se a lista está vazia. Caso a lista esteja vazia, o novo nó será o primeiro e o último da lista. Como o novo nó é o primeiro da lista, a referência `anterior` dele deve ser `NULL`. Além disso, a referência `anterior` do nó que era o primeiro da lista deve apontar para o novo nó. O novo nó passa a ser o primeiro da lista.

Inserção no final

Para inserir um novo nó no final da lista, basta criar um novo nó, atribuir o valor desejado a ele e percorrer a lista até encontrar o último nó.

```
void adicionar_final(int valor)
{
    struct Node *novo_node = (struct Node *)malloc(sizeof(struct Node));
    novo_node->data = valor;
    novo_node->proximo = NULL;
    if (inicio_da_lista == NULL)
    {
        novo_node->anterior = NULL;
        inicio_da_lista = novo_node;
    }
    else
    {
        struct Node *atual = inicio_da_lista;
        while (atual->proximo != NULL)
        {
            atual = atual->proximo;
        }
        atual->proximo = novo_node;
        novo_node->anterior = atual;
    }
}
```

Perceba que, ao adicionar um novo nó no final da lista, é necessário verificar se a lista está vazia. Caso a lista esteja vazia, o novo nó será o primeiro e o último da lista. Caso a lista não esteja vazia, é necessário percorrer a lista até encontrar o último nó. O último nó é aquele cuja referência `proximo` é `NULL`. O novo nó passa a ser o último da lista.

Remoção

A remoção de um nó de uma lista duplamente ligada pode ser feita no início, no final ou em qualquer posição da lista.

Remoção no início

Para remover o primeiro nó da lista, basta ajustar as referências do nó que era o segundo da lista e do nó que era o primeiro da lista.

```
void remover_inicio()
{
    struct Node *temp = inicio_da_lista;
    if (inicio_da_lista == NULL)
    {
        return;
```

```
    }
    inicio_da_lista = inicio_da_lista->proximo;
    inicio_da_lista->anterior = NULL;
    free(temp);
}
```

Perceba que, ao remover o primeiro nó da lista, é necessário verificar se a lista está vazia. Caso a lista esteja vazia, não é possível remover o primeiro nó. Caso a lista não esteja vazia, o segundo nó da lista passa a ser o primeiro. Além disso, a referência `anterior` do novo primeiro nó deve ser `NULL`.

Remoção no final

Para remover o último nó da lista, basta percorrer a lista até encontrar o penúltimo nó e ajustar a referência `proximo` dele.

```
void remover_final()
{
    struct Node *atual = inicio_da_lista;
    struct Node *anterior = NULL;
    while (atual->proximo != NULL)
    {
        anterior = atual;
        atual = atual->proximo;
    }
    if (anterior == NULL)
    {
        inicio_da_lista = NULL;
    }
    else
    {
        anterior->proximo = NULL;
    }
    free(atual);
}
```

Perceba que, ao remover o último nó da lista, é necessário verificar se a lista está vazia. Caso a lista esteja vazia, não é possível remover o último nó. Caso a lista não esteja vazia, é necessário percorrer a lista até encontrar o penúltimo nó. O penúltimo nó é aquele cuja referência `proximo` é o último nó.

Remoção em qualquer posição

Para remover um nó de qualquer posição da lista, basta percorrer a lista até encontrar o nó desejado e ajustar as referências do nó anterior e do nó posterior.

```
void remover_elemento(int valor)
{
    struct Node *atual = inicio_da_lista;
    struct Node *anterior = NULL;
    while (atual != NULL)
    {
        if (atual->data == valor)
        {
            if (anterior == NULL)
            {
                remover_inicio();
            }
            else
            {
                anterior->proximo = atual->proximo;
                free(atual);
            }
            return;
        }
        anterior = atual;
        atual = atual->proximo;
    }
}
```

Perceba que, ao remover um nó de qualquer posição da lista, é necessário percorrer a lista até encontrar o nó desejado. Caso o nó desejado seja o primeiro da lista, a função `remover_inicio` é chamada.

As funções de tamanho e impressão da lista são implementadas de maneira semelhante à lista simplesmente ligada.

```
void imprimir_lista()
{
    struct Node *atual = inicio_da_lista;

    printf(" INICIO --> ");
    while (atual != NULL)
    {
        printf("%d --> ", atual->data);
        atual = atual->proximo;
    }
    printf(" NULL\n");
}

int tamanho_da_lista()
```

```
{
    struct Node *atual = inicio_da_lista;
    int tamanho = 0;
    while (atual != NULL)
    {
        tamanho++;
        atual = atual->proximo;
    }
    return tamanho;
}

void imprimir_menu()
{
    printf("1. Adicionar elemento\n");
    printf("2. Adicionar elemento no final\n");
    printf("3. Remover elemento\n");
    printf("4. Remover elemento no final\n");
    printf("5. Remover elemento no inicio\n");
    printf("6. Imprimir lista\n");
    printf("7. Tamanho da lista\n");
    printf("8. Sair\n");
}
```

O programa principal é implementado da seguinte maneira:

```
int main()
{
    criar_lista();
    int opcao;
    int valor;
    do
    {
        imprimir_menu();
        printf("Digite a opcao desejada: ");
        scanf("%d", &opcao);
        switch (opcao)
        {
        case 1:
            printf("Digite o valor a ser adicionado: ");
            scanf("%d", &valor);
            adicionar_elemento(valor);
            break;
        case 2:
            printf("Digite o valor a ser adicionado: ");
            scanf("%d", &valor);
            adicionar_final(valor);
            break;
        case 3:
            printf("Digite o valor a ser removido: ");
            scanf("%d", &valor);
            remover_elemento(valor);
            break;
```

```
            case 4:
                remover_final();
                break;
            case 5:
                remover_inicio();
                break;
            case 6:
                imprimir_lista();
                break;
            case 7:
                printf("Tamanho da lista: %d\n", tamanho_da_lista());
                break;
            case 8:
                printf("Saindo...\n");
                break;
            default:
                printf("Opcao invalida\n");
                break;
            }
        } while (opcao != 8);
        return 0;
    }
```

Conclusão

Nesta aula, aprendemos a implementar uma lista duplamente ligada em C. A lista duplamente ligada é uma estrutura de dados linear que permite a inserção e remoção de elementos em qualquer posição da lista. A lista duplamente ligada é uma estrutura de dados dinâmica, ou seja, ela pode crescer ou diminuir de tamanho durante a execução do programa.

Referências

- Lista duplamente ligada
- Lista duplamente ligada em C
- Lista duplamente ligada em C
- Lista duplamente ligada em C