Tipo Abstrato de Dados (TAD), Lista Encadeada

1. Tipo Abstrato de Dados (TAD)

O TAD é definido como um modelo matemático ou uma abstração de dados que possibilita organizar e encapsular os dados e as operações (inserção, remoção, busca, etc.) realizadas neles. O TAD fornece uma interface pública para a interação com os dados, mas oculta os detalhes da implementação.

Isso permite que o programador se concentre na abstração dos dados e operações que podem ser realizadas neles, independentemente de como essa implementação será desenvolvida.

1.1. Por que utilizar TADs?

- Abstração
- Encapsulamento
- Manutenção
- Modularidade
- Reusabilidade

1.2. Exemplos de TADs:

- Árvores
- Dicionários
- Filas
- Listas
- Pilhas

Exemplo:

```
typedef struct lista Lista;
Lista* inserirNoInicio(Lista *inicio, int valor);
Lista* removerNo(Lista *inicio, int valor);
void imprimirLista(Lista *inicio);
void liberarLista(Lista *inicio);
```

Código completo:

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

// Definição do Tipo Abstrato de Dados (TAD) e dos Protótipos das funções do TAD
typedef struct lista Lista;
Lista* inserirNoInicio(Lista *inicio, int valor);
Lista* removerNo(Lista *inicio, int valor);
void imprimirLista(Lista *inicio);
void liberarLista(Lista *inicio);

// Definição da estrutura de uma lista
```

```c
struct lista {
  int nro;
  struct lista *proximo;
};

// Implementação das funções do TAD Lista
// Insere um elemento no início da lista
Lista* inserirNoInicio(Lista *inicio, int valor) {
  Lista *novoNo = (Lista*) malloc(sizeof(Lista));
  if (novoNo == NULL) {
      printf("Falha na alocação de memória! \n");
      exit(1);
  }
  novoNo->nro = valor;
  novoNo->proximo = inicio;
  return novoNo;
}

// Remove um elemento da lista
Lista* removerNo(Lista *inicio, int valor) {
  Lista *anterior = NULL; // ponteiro para o elemento anterior
  Lista *p = inicio;      // ponteiro para percorrer a lista

  // Procura o elemento na lista, armazenando o anterior
  while (p != NULL && p->nro != valor) {
    anterior = p;
    p = p->proximo;
  }

  // Verifica se encontrou o elemento
  if (p == NULL) {
    return inicio; // Não encontrou. Retorna a lista original
  }

  // Remove o elemento
  if (anterior == NULL) {
    inicio = p->proximo;  // Retira o elemento do início da lista
  } else {
    anterior->proximo = p->proximo;  // Retira o elemento do meio da lista
  }

  free(p);
```

```
  return inicio;
}

// Imprime os elementos da lista
void imprimirLista(Lista *inicio) {
  Lista *atual = inicio;
  while (atual != NULL) {
    printf("%d -> ", atual->nro);
    atual = atual->proximo;
  }
  printf("NULL \n");
}

// Libera a memória alocada pela lista
void liberarLista(Lista *inicio) {
  while (inicio != NULL) {
    Lista *aux = inicio;
    inicio = inicio->proximo;
    free(aux);
  }
}

int main() {
  Lista *minhalista = NULL;

  minhalista = inserirNoInicio(minhalista, 10);
  minhalista = inserirNoInicio(minhalista, 20);
  minhalista = inserirNoInicio(minhalista, 30);

  printf("Lista Encadeada: ");
  imprimirLista(minhalista);

  minhalista = removerNo(minhalista, 30);
  printf("Lista após a remoção do elemento 30: ");
  imprimirLista(minhalista);

  minhalista = removerNo(minhalista, 10);
  printf("Lista após a remoção do elemento 10: ");
  imprimirLista(minhalista);

  liberarLista(minhalista);

  return 0;
```

}

2. Exercícios

2.1. A implementação do exemplo de TAD foi organizada em único arquivo. Contudo, também é possível organizar o TAD em vários arquivos. Reorganize o código do exemplo em vários arquivos e apresente as vantagens e desvantagens de cada uma das abordagens.

2.2. Implemente uma lista encadeada para organizar uma lista de atividades. As tarefas devem ser inseridas no início da lista e removidas do final da lista. Além das operações de inserção e remoção, implemente a impressão das atividades.

Utilize TAD e Struct.

2.3. Implemente uma lista encadeada para organizar uma agenda de contatos. Devem ser implementadas as funcionalidades de inserção, remoção, busca e impressão dos contatos.

Utilize TAD e Struct.

2.4. Implemente uma lista encadeada para manipular uma lista de músicas (uma *playlist*). Devem ser implementadas as funcionalidades de inserção, remoção, busca, impressão e reproduzir próxima.

Utilize TAD e Struct.

Desafio:

- Realize um estudo sobre o conceito de lista duplamente encadeada. Após isso, implemente um exemplo que utilizando o conceito. Sugestão: Crie um programa que implemente as funcionalidades de “Repetir” e “Desfazer” a digitação em um texto.