Lista Linear, Lista Sequencial, Lista Encadeada

1. Lista Linear

Uma lista linear é uma estrutura de dados na qual os elementos estão organizados de maneira sequencial. Essas listas podem ser implementadas de várias formas diferentes. Iremos estudar duas opções: lista linear sequencial (ou lista sequencial) e lista simplesmente encadeada (ou lista encadeada).

1.1. Lista Sequencial

A lista sequencial é uma coleção de elementos armazenados em posições contíguas da memória. Cada elemento é acessado por meio de um índice (inicia em 0), que representa sua posição na lista. As listas sequenciais possuem um tamanho fixo determinado durante a sua criação e não podem ser alteradas dinamicamente.

Vantagens:

- O acesso aos elementos é rápido, pois é possível calcular sua posição na memória diretamente, com base no índice.
- A implementação de listas sequenciais é relativamente simples e eficiente em termos de memória.

Desvantagens:

- O tamanho da lista é estático, limitando sua utilização para casos nos quais a quantidade de elementos varia ao longo do tempo.
- A inserção e remoção de elementos no meio da lista requer a realocação de todos os elementos subsequentes, tornando o processo ineficiente.

Exemplo 1:

```
#include <stdio.h>

#define MAX_SIZE 3      // Tamanho da lista

int lista[MAX_SIZE];    // Declaração de uma lista sequencial
int tamanho_atual = 0;  // Variável para rastrear o tamanho atual da lista

// Função para adicionar um elemento ao final da lista
void inserirElemento(int elemento) {
  if (tamanho_atual < MAX_SIZE) {
    lista[tamanho_atual] = elemento;
    tamanho_atual++;
    printf("Elemento %d adicionado! \n", elemento);
  } else {
    printf("Lista completa. Não é possível adicionar mais elementos! \n");
  }
}
```

```
// Função para remover um elemento da lista por meio do índice
void removerElemento(int indice) {
  if (indice >= 0 && indice < tamanho_atual) {
    for (int i = indice; i < tamanho_atual - 1; i++) {
      lista[i] = lista[i + 1];
    }
    tamanho_atual--;
    printf("Elemento no índice %d removido com sucesso.\n", indice);
  } else {
    printf("Índice fora dos limites da lista. Nenhum elemento foi removido! \n");
  }
}

// Função para imprimir a lista
void imprimirLista() {
  if (tamanho_atual == 0) {
    printf("A lista está vazia! \n");
  } else {
    printf("Lista Sequencial: ");
    for (int i = 0; i < tamanho_atual; i++) {
      printf("%d ", lista[i]);
    }
    printf("\n");
  }
}

int main() {
  inserirElemento(1);
  inserirElemento(2);
  inserirElemento(3);
  inserirElemento(4);

  imprimirLista();
  removerElemento(2);
  imprimirLista();
  removerElemento(3);

  return 0;
}
```

1.2. Lista Encadeada

A lista encadeada é uma estrutura de dados que representa uma coleção de elementos de forma linear. Cada elemento, chamado de nó, contém dois campos principais: um valor e um

ponteiro para o próximo nó na lista. Isso permite a criação de uma sequencia de nós conectados, na qual o último nó aponta para NULL, indicando o fim da lista. Exemplo 2:

```
struct No {
  int valor;
  struct No *proximo;
};
```

Vantagens:

- Permite a criação de estruturas de dados de tamanho flexível, adaptando-se às necessidades do programa.
- As operações de inserção e remoção no início da lista são eficientes, o que é útil em cenários onde as mudanças frequentes na estrutura de dados são necessárias.
- A alocação dinâmica de memória possibilita um uso eficiente dos recursos de memória.

Desvantagens:

- Para acessar um elemento em uma posição específica da lista, é necessário percorrer a lista a partir do início, o que torna o acesso por índice ineficiente em comparação com arrays.
- Cada nó da lista requer espaço de memória adicional para armazenar os ponteiros, o que pode resultar em uso de memória ligeiramente maior em comparação com arrays.

Exemplo 3:

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

// Definição da estrutura do nó da lista
struct lista {
  int nro;
  struct lista *proximo;
};

typedef struct lista Lista;

// Função para inserir um número 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;
}
```

```c
// Função para remover o primeiro nó com o número informado
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;
}

// Função para imprimir a lista
void imprimirLista(Lista *inicio) {
  Lista *atual = inicio;
  while (atual != NULL) {
    printf("%d -> ", atual->nro);
    atual = atual->proximo;
  }
  printf("NULL \n");
}

int main() {
  Lista *inicio = NULL;

  // Inserir elementos na lista
```

```
    inicio = inserirNoInicio(inicio, 10);
    inicio = inserirNoInicio(inicio, 12);
    inicio = inserirNoInicio(inicio, 18);
    inicio = inserirNoInicio(inicio, 23);
    inicio = inserirNoInicio(inicio, 9);

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

    // Remover o elemento 12 da lista
    inicio = removerNo(inicio, 12);

    printf("Lista após a remoção do elemento 12: ");
    imprimirLista(inicio);

    /* // Exemplo de mais uma remoção
    inicio = removerNo(inicio, 9);
    printf("Lista após a remoção do elemento 9: ");
    imprimirLista(inicio);
    */

    // Liberar a memória alocada para os nós da lista
    while (inicio != NULL) {
      Lista *aux = inicio;
      inicio = inicio->proximo;
      free(aux);
    }

    return 0;
}
```

2. Exercícios

2.1. Ao alterar o código do exemplo “lista_sequencial.c”, removendo o trecho de código descrito a seguir das linhas iniciais para dentro da função main(), o programa apresenta erro de compilação. Ajuste o código para que o programa funcione com essa alteração.

```
int main() {
  int lista[MAX_SIZE];    // Declaração de uma lista sequencial
  int tamanho_atual = 0;  // Variável para rastrear o tamanho atual da lista
```

2.2. Adicione uma função para buscar elementos nas listas “sequencial” e “encadeada” apresentadas como exemplo.

2.3. Escreva duas versões (lista sequencial e encadeada) para um programa que controla a fila de um banco. O cliente deve ser chamado de acordo com o número da senha entregue (ordem crescente). Idosos (idade acima de 60 anos) possuem prioridade em relação aos demais clientes.

Observação: Uma única lista deve ser utilizada em cada uma das versões do programa.

2.4. Escreva duas versões (lista sequencial e encadeada) para um programa que simule o jogo de cartas 21.

2.5. Escreva um programa que permita a criação de um roteiro de viagem. Para a construção do roteiro, devem ser informados a cidade e a quantidade de dias que serão utilizados para visitá-la. O roteiro deve ser conter uma organização sequencial.

Caso de teste: Crie o seguinte roteiro: São Paulo (início: 0 dias) -> Bogotá (3 dias) -> Madrid (3 dias) -> Barcelona (2 dias) -> Londres (4 dias) -> Lisboa (1 dia) -> São Paulo (fim: 0 dias)

2.6. Implemente uma lista encadeada contendo os conteúdos da disciplina. Eles devem ser organizados e impressos de acordo com a ordem de dificuldade (mais difícil para mais fácil).

Desafio:

- Realize um estudo sobre o conceito de lista duplamente encadeada. Após isso, implemente um exemplo que utilize o conceito estudado.