Lista - Usada quando não se sabe o tamanho de inserções a serem realizadas, ou seja, quando não se sabe a quantidade de elementos a serem armazenados. Cada elemento representa um nó da lista.

- Lista x Arrays

- Inserção e remoção são mais fáceis em Listas encadeadas, enquanto em vetores é necessário movimentar todos os outros elementos seguintes ao elemento afetado.
- Para encontrar um termo em uma posição K, é mais fácil fazê-lo em um vetor (vet[K]), pois em uma lista encadeada será necessário percorrer K elementos.

Fila - Elementos sempre são inseridos no final e retirados do começo, como por exemplo uma fila de impressão, em que novas impressões são colocadas ao final da fila e retiradas do começo da fila. O primeiro que sai é o primeiro que entrou (FIFO – First in, first out).

Pilha - Empilhamento de elementos, em que as inserções ocorrem no topo da lista e também são retiradas do topo. O primeiro que sai é o último que entrou (LIFO – Last in, first out).

Códigos

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

typedef struct lista {
   int valor;
   struct lista *prox;
}Lista;

//função para a exibição da lista
void exibe(Lista *no){
   no = no->prox; // nó é deslocado para o endereço que o cabeça aponta na 1° iteração
   printf("--------Lista--------\n");
   while(no != NULL) {
      printf("%d ", no->valor);
      no = no->prox;
   }
   printf("\n");
}

//função que cria um novo elemento para a lista(alocando espaço de memória)
Lista* criaNovo(int val) {
   Lista *elem;
   elem = (Lista*) malloc(sizeof(Lista));
   if(elem == NULL) {
      printf("Erro ao alocar memória!\n");
   } else {
```

```
        elem->valor = val;
        elem->prox = NULL;
    }
    return elem;
}

//inserção no início (cria um novo elemento contendo o valor passado e o adiciona à lista encadeada)
//função NÃO recursiva
void inserirInicio(Lista *lista, int valor) {
    Lista *novo = (Lista*) malloc(sizeof(Lista));
    if (novo == NULL) {
        printf("Erro ao alocar memória!\n");
    } else {
        novo->valor = valor;
        if(lista->prox == NULL) {
            novo->prox = NULL;
            lista->prox = novo;
        } else {
            novo->prox = lista->prox;
            lista->prox = novo;
        }
    }
}

//função que busca um valor na lista e retorna o elemento caso o valor exista na lista
Lista* buscaValor(Lista *pLista, int valor) {
    while(pLista != NULL) {
        if( pLista->valor == valor){
            return (pLista);
        } else {
            pLista = pLista->prox;
        }
    }
    return NULL;
}

//função de inserção no final da lista recursivamente
void inserirFim(Lista *lista, Lista *elem) {
    if(lista->prox == NULL) { //caso base
        lista->prox = elem;
    } else { //caso geral
        inserirFim(lista->prox, elem);
    }
```

```
}

// função de remoção no final da lista recursivamente
void removeFim(Lista *lista) {
   if(lista->prox->prox == NULL) { //caso base
      Lista *lixo;
      lixo = lista->prox;
      lista->prox = NULL;
      free(lixo);
   } else {
      removeFim(lista->prox); //caso geral
   }
}

void removeInicio(Lista *lista) {
   Lista *lixo;
   lixo = lista->prox;
   lista->prox = lista->prox->prox;
   free(lixo);
}

int main() {
   int opcao, valor;
   Lista *cabeca, *elem;
   cabeca = (Lista*) malloc(sizeof(Lista));
   cabeca->prox = NULL;

   do {
      printf("\nMenu de Opções:\n");
      printf("\t0 - Sair\n");
      printf("\t1 - Inserir no Início não Recursivamente\n");
      printf("\t2 - Inserir no Final Recursivamente\n");
      printf("\t3 - Remover do Final Recursivamente\n");
      printf("\t4 - Imprimir\n");
      scanf("%d", &opcao);

      switch(opcao) {
         case 1:
            printf("Digite um valor: ");
            scanf("%d", &valor);
            inserirInicio(cabeca, valor);
            break;
         case 2:
            printf("Digite um valor: ");
```

```
            scanf("%d", &valor);
            elem = criaNovo(valor);
            inserirFim(cabeca, elem);
            break;
        case 3:
            printf("Esvaziando a pilha...\n");
            removeFim(cabeca);
            break;
        case 4:
            exibe(cabeca);
            break;
        default:
            if(opcao!=0)
                printf("Opção inválida!\n");
      }
   } while(opca
o != 0);

   return 0;
}
```

Roteiro

-Ponteiros
-Struct: variáveis e ponteiros
-Lista Encadeada
-Inserção e Remoção

```
void inserirInicio (Lista *lista, int valor) {
   Lista *novo = (Lista*) malloc(sizeof(Lista));
   if (novo == NULL) {
      printf("Erro ao alocar memória!\n");
   } else {
      novo->valor = valor;
      if(lista->prox == NULL) {
         novo->prox = NULL;
         lista->prox = novo;
      } else {
         novo->prox = lista->prox;
         lista->prox = novo;
      }
   }
}
```

```
//função de inserção no final da lista recursivamente
void inserirFim(Lista *lista, Lista *elem) {
   if(lista->prox == NULL) { //caso base
      lista->prox = elem;
   } else { //caso geral
      inserirFim(lista->prox, elem);
   }
}

// função de remoção no final da lista recursivamente
void removeFim(Lista *lista) {
   if(lista->prox->prox == NULL) { //caso base
      Lista *lixo;
      lixo = lista->prox;
      lista->prox = NULL;
      free(lixo);
   } else {
      removeFim(lista->prox); //caso geral
   }
}

void removeInicio(Lista *lista) {
   Lista *lixo;
   lixo = lista->prox;
   lista->prox = lista->prox->prox;
   free(lixo);
}
```