ESTRUTURA DE DADOS – Revisão

(Lista[List]; Fila/Deque[Queue-FIFO]; Pilha[Stack-FILO/LIFO])

Estruturas de Dados Estática

```
#define MAX 50
typedef int TIPOCHAVE;
typedef struct{
  TIPOCHAVE chave;
  // Outros Campos;
} REGISTRO;
```

```
typedef struct {
  REGISTRO A[MAX];
  int nroElem;
} LISTASequencialEstatica;
```

```
typedef struct {
  REGISTRO A[MAX];
  int topo;
} PILHAEstatica;
```

```
typedef struct {
  REGISTRO A[MAX];
  int inicio;
  int nroElem;
} FILAEstatica;
```

```
typedef struct {
  ELEMENTO A[MAX];
  int inicio;
  int dispo;
} LISTAEncadeadaEstatica;
```

```
typedef struct {
  REGISTRO reg;
  int prox;
} ELEMENTO;
```

Documentação:

Chave: é o elemento (número inteiro);
A[MAX]: Arranjo de elementos;
nroElem: quantidade de elementos na estrutura de dados
topo: índice(posição) do elemento que está no TOPO da pilha
inicio: índice(posição física) do primeiro elemento da LISTA
dispo: índice(posição física) do elemento disponível na LISTA
prox: índice (posição) de seu elemento sucessor na estrutura de dados (arranjo)

Nota: Lista Linear: Sequencial; Encadeada; Duplamente Encadeada; Circular.

Estrutura de Dados Listas Dinâmica

```
typedef int TIPOCHAVE;
typedef struct{
    TIPOCHAVE chave;
    // Outros Campos;
} REGISTRO;
```

```
typedef struct aux {
    REGISTRO reg;
    struct aux* prox;
} ELEMENTO;
typedef ELEMENTO* PONT;
```

```
typedef struct {
  PONT inicio;
} LISTAEncadeadaDinamica;
```

```
typedef struct {
  PONT cabeca;
} LISTACircularDinamica;
```

```
typedef struct {
  PONT inicio;
} LISTADuplamenteEncadeadaDinamica;
```

Documentação:

Chave: é o elemento (número inteiro);
reg: elemento na estrutura de dados ELEMENTO;
nroElem: quantidade de elementos na LISTA
topo: endereço de memória onde encontra-se o elemento na LISTA
cabeca = endereço de memória do nó cabeca
inicio: endereço de memória onde inicia o elemento na LISTA
nó cabeca = nó cabeça para controlar a LISTA.

Estrutura de Dados Pilha Dinâmica

```
typedef int TIPOCHAVE;
typedef struct{
    TIPOCHAVE chave;
    // Outros Campos;
} REGISTRO;
```

```
typedef struct aux {
    REGISTRO reg;
    struct aux* prox;
} ELEMENTO;
typedef ELEMENTO* PONT;
```

```
typedef struct {
  PONT topo;
} PILHADinamica;
```

Documentação:

Chave: é o elemento (número inteiro);
reg: elemento na estrutura de dados ELEMENTO;
prox: endereço de memória do elemento na PILHA
topo: endereço de memória onde encontra-se o elemento na PILHA

Estrutura de Dados Fila Dinâmica

```
typedef int TIPOCHAVE;
typedef struct{
    TIPOCHAVE chave;
} REGISTRO;
```

```
typedef struct aux {
    REGISTRO reg;
    struct aux* prox;
} ELEMENTO,* PONT;
```

```
typedef struct {
  PONT inicio;
  PONT fim;
} FILADinamica;
```

Documentação:

Chave: é o elemento (número inteiro);
reg: elemento na estrutura de dados ELEMENTO;
topo: endereço de memória onde encontra-se o elemento na FILA
inicio: endereço de memória onde inicia o elemento na FILA

Funções (Métodos):

1. Inicializar a estrutura de dados
2. Retornar a quantidade de elementos válidos
3. Exibir os elementos da estrutura de dados
4. Buscar por um elemento na estrutura de dados
5. Inserir elementos na estrutura de dados **(Pilha: início); (Fila: no fim)**
6. Excluir elementos da estrutura de dados **(Pilha: início); (Fila: no início)**
7. Reinicializar a estrutura de dados
8. Salvar a estrutura de dados
9. Carregar estrutura de dados

Controles: Verificar se a estrutura de dados está vazia.

Tipos de Busca: Sequencial; Sentinela; Binária (Estrutura Ordenada).

Nota: Inserção na Pilha (método chamado de **push - empilhar**);
Exclusão na Pilha (método chamado de **pop - desempilhar**);

1. Inicializar a estrutura de dados

1. Inicializar/Reinicializar estrutura de dados Estática

```
void inicializarLista(LISTA* lista) {
lista.nroElem = 0;
}
```

2. Inicializar estrutura de dados Dinâmica

```
void inicializarLista(LISTA* lista){
lista->inicio = NULL;
}
```

3. Reinicializar estrutura de dados Dinâmica

```
void reinicializarLista(LISTA* lista) {
PONT end = lista->inicio;
while (end != NULL) {
PONT apagar = end;
end = end->prox;
free(apagar);
}
lista->inicio = NULL;
}
```

2. Retornar a quantidade de elementos válidos

1. Retornar número de elementos da estrutura de dados Estática

```
int tamanho(LISTA* lista) {
return lista->nroElem;
}
```

2. Retornar número de elementos da estrutura de dados Dinâmica

```
int tamanho(LISTA* lista) {
PONT end = lista->inicio;
int tam = 0;
while (end != NULL) {
tam++;
end = end->prox;
}
return tam;
}
```

3. Exibir os elementos da estrutura de dados

1. Exibir elementos da estrutura de dados Estática

```
void exibirLista(LISTA* lista){
int i;
printf("Lista: \" ");
for (i=0; i < lista->nroElem; i++)
printf("%i ", lista->A[i].chave);
printf("\"\n");
}
```

2. Exibir elementos da estrutura de dados Dinâmica

```
void exibirLista(LISTA* lista){
PONT end = lista->inicio;
printf("Lista: \" ");
while (end != NULL) {
printf("%i ", end->reg.chave);
end = end->prox;
}
printf("\"\n");
}
```

4. Inserir elementos na estrutura de dados (Pilha: início); (Fila: no fim)

1. Inserir elemento na estrutura de dados Estática

```
bool inserirElemLista(LISTA* lista, REGISTRO reg, int i){
int j;
if ((lista->nroElem == MAX) || (i < 0) || (i > lista->nroElem))
return false;
for (j = lista->nroElem; j > i; j--) lista->A[j] = lista->A[j-1];
lista->A[i] = reg;
lista->nroElem++;
return true;
}
```

2. Inserir elemento da estrutura de dados Dinâmica

```
bool inserirElemListaOrd(LISTA* lista, REGISTRO reg) {
TIPOCHAVE ch = reg.chave;
PONT ant, i;
i = buscaSequencialExc(lista,ch,&ant);
if (i != NULL) return false;
i = (PONT) malloc(sizeof(ELEMENTO));
i->reg = reg;
if (ant == NULL) {
i->prox = lista->inicio;
lista->inicio = i;
} else {
i->prox = ant->prox;
ant->prox = i;
}
return true;    }
```

5. Excluir elementos da estrutura de dados (Pilha: início); (Fila: no início)

1. Excluir elemento da estrutura de dados Estática

```
bool excluirElemLista(TIPOCHAVE ch, LISTA* lista) {
int pos, j;
pos = buscaSequencial(lista,ch);
if(pos == -1) return false;
for(j = pos; j < lista->nroElem-1; j++)
lista->A[j] = lista->A[j+1];
lista->nroElem--;
return true;
}
```

2. Excluir elemento da estrutura de dados Dinâmica

```
bool excluirElemLista(LISTA* lista, TIPOCHAVE ch) {
PONT ant, i;
i = buscaSequencialExc(lista,ch,&ant);
if (i == NULL) return false;
if (ant == NULL) lista->inicio = i->prox;
else ant->prox = i->prox;
free(i);
return true;
}
```

1. **Busca Sequencial Estática**

```
int buscaSequencial(LISTA* lista, TIPOCHAVE ch) {
int i = 0;
while (i < lista->nroElem){
if(ch == lista->A[i].chave) return i;
else i++;
}
return -1;
}
```

2. **Busca Sequencial Dinâmica**

```
PONT buscaSequencial(LISTA* lista, TIPOCHAVE ch) {
PONT pos = lista->inicio;
while (pos != NULL) {
if (pos->reg.chave == ch) return pos;
pos = pos->prox;
}
return NULL;
}
```

3. **Busca Sequencial Antecessor_Sucessor Dinâmica**

```
PONT buscaSequencialExc(LISTA* lista, TIPOCHAVE ch, PONT* ant){
*ant = NULL;
PONT atual = lista->inicio;
while ((atual != NULL) && (atual->reg.chave<ch)) {
*ant = atual;
atual = atual->prox;
}
if ((atual != NULL) && (atual->reg.chave == ch)) return atual;
return NULL;
}
```

4. Busca Sequencial Ordenada Estática

```
int buscaSequencialOrd(LISTA* lista, TIPOCHAVE ch) {
int i = lista->inicio;
while (i != -1 && lista->A[i].reg.chave < ch)
i = lista->A[i].prox;
if (i != -1 && lista->A[i].reg.chave == ch)
return i;
else return -1;
}
```

5. Busca Sequencial Ordenada Dinâmica

```
// lista ordenada pelos valores das chaves dos registros
PONT buscaSeqOrd(LISTA* lista, TIPOCHAVE ch) {
PONT pos = lista->inicio;
while (pos != NULL && pos->reg.chave < ch) pos = pos->prox;
if (pos != NULL && pos->reg.chave == ch) return pos;
return NULL;
}
```

6. Busca Sentinela

```
int buscaSentinela(LISTA* lista, TIPOCHAVE ch) {
int i = 0;
lista->A[lista->nroElem].chave = ch;
while(lista->A[i].chave != ch) i++;
if (i == lista->nroElem) return -1;
else return i;
}
```

Nota: Criar a lista com uma posição extra (um registro a mais) para garantir que haverá espaço para o sentinela. Essa posição extra nunca terá um registro válido.

7. **Busca Binária**

```
int buscaBinaria(LISTA* lista, TIPOCHAVE ch) {
int esq, dir, meio;
esq = 0;
dir = lista->nroElem-1;
while(esq <= dir) {
meio = ((esq + dir) / 2);
if(lista->A[meio].chave == ch) return meio;
else {
if(lista->A[meio].chave < ch) esq = meio + 1;
else dir = meio - 1;
}
 }
 }
```

1. **Inclusão na PILHA Dinâmica**

```
bool inserirElemPilha(PILHA* p, REGISTRO reg) {
PONT novo = (PONT) malloc(sizeof(ELEMENTO));
novo->reg = reg;
novo->prox = p->topo;
p->topo = novo;
return true;
}
```

2. **Exclusão na PILHA Dinâmica**

```
bool excluirElemPilha(PILHA* p, REGISTRO* reg) {
if ( p->topo == NULL) return false;
*reg = p->topo->reg;
PONT apagar = p->topo;
p->topo = p->topo->prox;
free(apagar);
return true; }
```

3. **Inclusão na FILA Dinâmica**

```
bool inserirNaFila(FILA* f,REGISTRO reg) {
PONT novo = (PONT) malloc(sizeof(ELEMENTO));
novo->reg = reg;
novo->prox = NULL;
if (f->inicio==NULL) f->inicio = novo;
else f->fim->prox = novo;
f->fim = novo;
return true;
}
```

4. **Exclusão na FILA Dinâmica**

```
bool excluirDaFila(FILA* f, REGISTRO* reg) {
if (f->inicio==NULL) return false;
*reg = f->inicio->reg;
PONT apagar = f->inicio;
f->inicio = f->inicio->prox;
free(apagar);
if (f->inicio == NULL) f->fim = NULL;
return true;
}
```

EXERCÍCIOS TAD

1. Escreva uma TAD de **lista** de inteiros ordenada simplesmente encadeada com as seguintes operações:

   a) Verificar se um número pertence lista;
   b) Inserir um novo elemento na lista mantendo a ordem;
   c) Remover um elemento da lista;
   d) Imprimir os valores da lista;
   e) Copiar uma lista l1 para uma lista l2;

2. Utilizando somente operações de empilhar e desempilhar, escreva um programa que remove um item com chave c fornecida pelo usuário da **pilha**. Ao final da execução da função, a pilha deve ser igual à original, exceto pela ausência do item removido.

3. Escreva um programa que simule o controle de uma pista de decolagem de aviões em um aeroporto. Neste programa, o usuário deve ser capaz de realizar as seguintes tarefas:

   a) Listar o número de aviões aguardando na **fila** de decolagem;
   b) Autorizar a decolagem do primeiro avião da fila;
   c) Adicionar um avião à fila de espera;
   d) Listar todos os aviões na fila de espera;
   e) Listar as características do primeiro avião da fila.

   Considere que os aviões possuem um nome e um número inteiro como identificador. Adicione outras características conforme achar necessário.