TAD Vetor

Desenvolver um tipo para manipular uma sequência de números inteiros.

- Os números podem se repetir no conjunto
- A sequência em que os números são inseridos precisa ser preservada.

Diversas operações deverão ser permitidas nesse conjunto, tais como inserção, remoção, substituição, busca, etc.

Dados

Quais dados serão manipulados?

- Uma sequência de números inteiros

Operações

Quais operações serão necessárias para manipular os dados?

1. Criar uma nova sequência
2. Inserir um elemento no final da sequência
3. Inserir um elemento em qualquer posição da sequência
4. Substituir um elemento em uma determinada posição
5. Remover um elemento de uma determinada posição
6. Remover um elemento de acordo com o seu valor. Caso haja valores repetidos remover apenas a primeira ocorrência.
7. Recuperar quantos elementos estão armazenados na sequência.
8. Recuperar qual elemento está armazenado em uma determinada posição
9. Recuperar a posição em que um determinado elemento está armazenado. Caso haja valores repetidos, recuperar a primeira ocorrência.
10. Imprimir a sequência
11. Destruir a sequência. Desalocar da memória
12. Recuperar a sequência na forma de string

Especificação

1. Definição de uma struct para encapsular a organização dos dados
2. Transformar cada funcionalidade em um protótipo de função
    - Para auxiliar a definição dos protótipos das funções, crie casos de testes utilizando as funções como se elas estivessem prontas.

tad_vetor.h

```c
#ifndef _TAD_VETOR_
#define _TAD_VETOR_

#include <stdlib.h>
#include <stdbool.h>

/*****************************************
 * DADOS
 *****************************************/
typedef struct vetor Vetor;

/*****************************************
 * OPERAÇÕES
 *****************************************/

// 1. Criar uma nova sequência
/**
 * Cria e inicializa a struct Vetor.
 * RETORNO: endereço da struct Vetor criada e inicializada na memória HEAP
 */
Vetor* vet_criar(int tam);

// 2. Inserir um elemento no final da sequência
/**
 * Insere o elemento na última posição do vetor.
 * Parâmetro v: Ponteiro para a struct Vetor em que o elemento será inserido.
 * Parâmetro elemento: Elemento a ser inserido.
 * RETORNO: true se a inserção foi realizada com sucesso e false caso contrário
 */
bool vet_anexar(Vetor* v, int elemento);
```

```c
// 3. Inserir um elemento em qualquer posição da sequência
/**
 * Insere um elemento em uma determinada posição.
 * Parâmetro v: Ponteiro para a struct Vetor em que o elemento será inserido.
 * Parâmetro elemento: Elemento a ser inserido.
 * Parâmetro posicao: Posição em que o elemento deve ser inserido.
 * RETORNO: true se a inserção foi realizada com sucesso e false caso contrário
 */
bool vet_inserir(Vetor* v, int elemento, int posicao);

// 4. Substituir um elemento em uma determinada posição
/**
 * Substitui o valor de uma determinada posição do Vetor.
 * Parâmetro v: Ponteiro para a struct Vetor.
 * Parâmetro posicao: Posição a ser alterada.
 * Parâmetro novoElemento: elemento a ser atribuido na posição.
 * RETORNO: true se a alteração foi realizada com sucesso e false caso contrário
 */
bool vet_substituir(Vetor* v, int posicao, int novoElemento);

// 5. Remover um elemento de uma determinada posição
/**
 * USANDO A ESTRATÉGIA DO SCANF
 * Remove o elemento de uma determinada posição do vetor .
 * Parâmetro v: Ponteiro para a struct Vetor.
 * Parâmetro posicao: posição a ser removida.
 * Parâmetro endereco: endereço a ser utilizado para a copiar o valor do elemento removido.
 * RETORNO: true se a inserção foi realizada com sucesso e false caso contrário
 */
bool vet_removerPosicao(Vetor* v, int posicao, int* endereco);
```

```c
// 6. Remover um elemento de acordo com o seu valor. Caso haja valores repetidos remover apenas a primeira ocorrência.
/**
 * Remove um determinado elemento do vetor .
 * Parâmetro v: Ponteiro para a struct Vetor.
 * Parâmetro elemento: elemento a ser removido.
 * RETORNO: posição do elemento removido. Caso o elemento não seja encontrado, a função deve devolver -1
 */
int vet_removerElemento(Vetor* v, int elemento);

// 7. Recuperar quantos elementos estão armazenados na sequência.
/**
 * Devolve a quantidade de elementos do vetor.
 * Parâmetro v: Ponteiro para a struct Vetor.
 * RETORNO: quantidade de elementos do vetor
 */
int vet_tamanho(Vetor* v);

// 8. Recuperar qual elemento está armazenado em uma determinada posição
/**
 * USANDO A ESTRATÉGIA DO SCANF
 * Pesquisa o elemento armazenado em uma determinada posição do Vetor.
 * Parâmetro v: Ponteiro para a struct Vetor.
 * Parâmetro posicao: posicao a ser encontrada.
 * Parâmetro saida: Endereço de memória onde a função deve armazenar o elemento encontrado.
 * RETORNO: Se a posição for válida, realiza a cópia no endereço recebido por parâmetro SAIDA e devolve true.
 *          Caso contrário, devolve false
 */
bool vet_elemento(Vetor* v, int posicao, int* saida);
```

```c
// 9. Recuperar a posição em que um determinado elemento está armazenado. Caso haja valores repetidos, recuperar a primeira ocorrência.
/**
 * Pesquisa a posição de um determinado elemento no Vetor.
 * Parâmetro v: Ponteiro para a struct Vetor.
 * Parâmetro elemento: elemento a ser procurado.
 * RETORNO: Se encontrado, devolve a posição do elemento no vetor; caso contrário devolve -1
 */
int vet_posicao(Vetor* v, int elemento);

// 10. Imprimir a sequência
/**
 * Imprimir os elementos do vetor
 * Parâmetro v: Ponteiro para a struct Vetor.
 */
int vet_imprimir(Vetor* v);

// 11. Desalocar
/**
 * Destruir/Desalocar/liberar o vetor na memória HEAP
 * Parâmetro v: Endereço da variável que armazena o ponteiro para a struct Vetor (ponteiro de ponteiro).
 */
void vet_desalocar(Vetor** endVetor);

// 12. Devolve o vetor na forma de String
/**
 * Escreve no endereço recebido por parâmetro uma versão string do vetor
 * Parâmetro v: Ponteiro para a struct Vetor.
 * Parâmetro endereco: endereço da região de memória onde a função deverá copiar os caracteres.
 * RETORNO: true se a cópia foi realizada com sucesso e false caso contrário
 */
bool vet_toString(Vetor* v, char* enderecoString);
```

```c
#endif
```

Utilização

Desenvolva os casos de teste simulando a utilização das funções. Depois de implementado, utilize esse arquivo para validar sua implementação

```c
//main.c
#include<stdio.h>
#include<stdlib.h>
#include "tad_vetor.h"

int main(){


}
```

Implementação

Implemente as funções descritas no tad_vetor.h

```c
//tad_vetor.c
#include "tad_vetor.h"

/*****************************************
 * Especificação dos dados
 *****************************************/
struct vetor{
    int* vet;
    int tam;
    int qtd;
};

/*****************************************
 * Implementação das funções
 *****************************************/
Vetor* vet_criar(int tam);
bool vet_anexar(Vetor* v, int elemento);
bool vet_inserir(Vetor* v, int elemento, int posicao);
bool vet_substituir(Vetor* v, int posicao, int novoElemento);
bool vet_removerPosicao(Vetor* v, int posicao, int* endereco);
int vet_removerElemento(Vetor* v, int elemento);
int vet_tamanho(Vetor* v);
bool vet_elemento(Vetor* v, int posicao, int* saida);
int vet_posicao(Vetor* v, int elemento);
void vet_imprimir(Vetor* v);
void vet_desalocar(Vetor** endVetor);
bool vet_toString(Vetor* v, char* saida);
```

Automatização da compilação

Crie o arquivo makefile para automatizar a compilação.

makefile

```
all:

run:
```