```
/*---------------------------------------------*/
/*          2º Exercicio-Programa              */
/*          Integrantes da Equipe              */
/*          Herculano Gonçalves Santos         */
/*          Lucas Diego Rebouças Rocha         */
/*          Thiago Duarte Medeiros             */
/*---------------------------------------------*/


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

/*Declaração das Estruturas de Dados*/


/* @brief       Estrutura de Dados NodoAVL
*               Estrutura representando um nodo de uma árvore AVL
*               composta por um inteiro (info) que guarda o valor do nodo,
*               dois ponteiros (esq e dir) do tipo NodoAVL, representando seus
*               filhos esquerdo e direito respectivamente
*/
typedef struct Nodo{

        int info;
        struct Nodo *esq;
        struct Nodo *dir;

}NodoAVL;
```

```
/* @brief       Estrutura de Dados TNodo
*            Estrutura representando um nodo de lista encadeada
*            composta por um ponteiro (info) que guarda o valor de um
NodoAVL,
*            um ponteiro (pro) do tipo TNodo, representando seu
*            proximo nodo
*/
struct nodo {

   NodoAVL *val;
   struct nodo *pro;
};
typedef struct nodo TNodo;

/*Fim da declaração das Estruturas de Dados*/

/
**************************************************************************
*/

/*Declaração das Funções auxiliares*/

/* @brief       Função inicializeLI
*            Função para a inicialização de uma lista ligada
*
*  @param       lis - um ponteiro para um ponteiro de uma lista ligada
*
*  @return      void
*/
```

```
void inicializeLI(TNodo **lis)
{
   *lis = NULL;
}

/* @brief       Função estaVaziaLI
*              Função para a verificação se a lista está vazia
*
*  @param       lis - um ponteiro para uma lista ligada
*
*  @return      1 - Se a lista está vazia
*               0 - se a lista não está vazia
*/
int estaVaziaLI(TNodo *lis)
{
   return lis == NULL;
}

/* @brief       Função insiraNoFinalLI
*              Função para a inserção de um valor em uma lista ligada
*
*  @param       lis - um ponteiro para um ponteiro de uma lista ligada
*               val - um ponteiro (NodoAVL) para um nodo de uma árvore AVL
*
*  @return      void
*/
void insiraNoFinalLI(TNodo **lis, NodoAVL *val)
{
```

```
    TNodo *nn = (TNodo *) malloc(sizeof(TNodo));
    nn->val = val;
    nn->pro = NULL;
    if(estaVaziaLI(*lis))
        *lis = nn;
    else
    {
        TNodo *p = *lis;
        while(p->pro != NULL)
            p = p->pro;
        p->pro = nn;
    }
}

/* @brief       Função removaDoInicio
*               Função para a remoção de um valor no inicio de uma lista ligada
*
*  @param       lis - um ponteiro para um ponteiro de uma lista ligada
*
*  @return      res - um ponteiro (NodoAVL) para um nodo de uma árvore AVL
*/
NodoAVL *removaDoInicio(TNodo **lis)
{
    TNodo *p = *lis;
    NodoAVL *res = p->val;
    *lis = p->pro;
    free(p);
    return res;
```

```
}


/* @brief       Função Vazia
*              Função para a verificação se o nodo é vazio
*
*  @param       nodo - um ponteiro para um nodo de uma arvore AVL
*
*  @return      1 - Se o nodo é vazio
*               0 - se o nodo é não vazio
*/
int estaVaziaAVL(NodoAVL *nodo){
   return nodo == NULL;
}

/* @brief       Função maior
*              Função auxiliar para a verificação de valores
*
*  @param       num_1 - numero inteiro
*               num_2 - numero inteiro
*
*  @return      o maior valor dentre num_1 e num_2
*/
int maior(int num_1, int num_2)
{
   if(num_1 > num_2) return num_1;
   else    return num_2;
}
```

```
/*Fim da declaração das Funções auxiliares*/

/
**********************************************************************************
*/

/* Declaração da Funções de manipulação da árvore */

/* @brief       Função num_elementos
*
*  @param       raiz - um ponteiro para a raiz de uma arvore AVL
*
*  @return      o numero de elementos desta arvore
*/
int num_elementos(NodoAVL *raiz){

   if(raiz == NULL){
         return 0;
   }

   return 1 + num_elementos(raiz->esq) + num_elementos(raiz->dir);
}



/* @brief       Função maior_elemento
*
*  @param       raiz - um ponteiro para a raiz de uma arvore AVL
*
```

```
*  @return      o valor do maior elemento contido na arvore
*/
int maior_elemento(NodoAVL *raiz){

    if(raiz == NULL){
            return 0;
    }

    NodoAVL *aux = raiz;

    while(aux->dir != NULL){
         aux = aux->dir;
    }

    return aux->info;

}

/* @brief       Função menor_elemento
*
*  @param       raiz - um ponteiro para a raiz de uma arvore AVL
*
*  @return      o valor do menor elemento contido na arvore
*/
int menor_elemento(NodoAVL *raiz){

    if(raiz == NULL){
            return 0;
```

```
    }

    NodoAVL *aux = raiz;

    while(aux->esq != NULL){
         aux = aux->esq;
    }

    return aux->info;

}



/* @brief       Função altura
*
*  @param       raiz - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      um inteiro representando o tamanho da arvore
*/
int altura(NodoAVL *raiz){

    if(raiz == NULL){
           return 0;
    }
    else{
        return 1 + maior(altura(raiz->esq), altura(raiz->dir));
    }
}
```

```
/* @brief       Função destroi_arvore
*              Destroi a arvore desalocando todos os seus nodos da memoria
*              e tornando a raiz nula
*
*  @param       raiz - um ponteiro (NodoAVL) para um ponteiro para a raiz de uma arvore AVL
*
*  @return      void
*/
void destroi_arvore(NodoAVL **raiz){

    if((*raiz) != NULL){
        destroi_arvore(&(*raiz)->dir);
        destroi_arvore(&(*raiz)->esq);
        free(raiz);
        *raiz=NULL;
    }

}

/
*****************************************************************************
*/

//Funções de Rotação da árvore

/* @brief       Função rotateR
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
```

```
*
*  @return      um nodo (NodoAVL) para uma arvore AVL desta ter sofrido
*               uma rotação simples a direita
*/
NodoAVL *rotateR( NodoAVL *nodo ){

      NodoAVL *aux;

      aux = nodo->esq;
      nodo->esq = aux->dir;
      aux->dir = nodo;

      return aux;
}

/* @brief       Função rotateL
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      um nodo (NodoAVL) para uma arvore AVL desta ter sofrido
*               uma rotação simples a esquerda
*/
NodoAVL *rotateL( NodoAVL *nodo ){

       NodoAVL *aux;

       aux = nodo->dir;
       nodo->dir = aux->esq;
```

```
	aux->esq = nodo;

	return aux;
}

/* @brief	Função rotateLR
*
*  @param	nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return	um nodo (NodoAVL) para uma arvore AVL desta ter sofrido
*		uma rotação dupla do tipo esquerda-direita
*/
NodoAVL *rotateLR( NodoAVL *nodo ){

	nodo->esq = rotateL( nodo->esq );

	return rotateR( nodo );
}

/* @brief	Função rotateRL
*
*  @param	nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return	um nodo (NodoAVL) para uma arvore AVL desta ter sofrido
*		uma rotação dupla do tipo direita-esquerda
*/
NodoAVL *rotateRL( NodoAVL *nodo )
{
```

```
        nodo->dir = rotateR( nodo->dir );

        return rotateL( nodo );
}




/* @brief       Função busca_Valor
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*               valor - um inteiro
*
*  @return      1 - caso o valor já exista na árvore
*               0 - caso contrário
*/
int busca_Valor(NodoAVL *nodo, int valor){

    if(nodo == NULL){
            return 0;
    }
    else if(nodo->info == valor){
            return 1;
    }
    else if(nodo->info < valor){
        return busca_Valor(nodo->dir, valor);
    }
    else{
        return busca_Valor(nodo->esq, valor);
```

```
    }

}

/
*****************************************************************************
*/

//Funções de percurso em Arvore

/* @brief       Função emOrdem
*               Apresenta os elementos da árvore percorrendo-a em ordem
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      void
*/

void emOrdem(NodoAVL *nodo)
{
    if(!estaVaziaAVL(nodo))
    {
        emOrdem(nodo->esq);
        printf(" %d |",nodo->info);
        emOrdem(nodo->dir);
    }
}

/* @brief       Função preOrdem
```

```
*              Apresenta os elementos da árvore percorrendo-a em pre ordem
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      void
*/
void preOrdem(NodoAVL *nodo)
{
   if(!estaVaziaAVL(nodo))
   {
      printf(" %d |",nodo->info);
      if(nodo->esq != NULL)
         preOrdem(nodo->esq);
      if(nodo->dir != NULL)
         preOrdem(nodo->dir);
   }
}

/* @brief       Função posOrdem
*              Apresenta os elementos da árvore percorrendo-a em pos ordem
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      void
*/
void posOrdem(NodoAVL *nodo)
{
   if(!estaVaziaAVL(nodo))
```

```
    {
        posOrdem(nodo->esq);
        posOrdem(nodo->dir);
        printf(" %d |",nodo->info);
    }
}

/* @brief       Função emLargura
*               Apresenta os elementos da árvore percorrendo-a em largura
*
*  @param       nodo - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      void
*/
void emLargura(NodoAVL *raiz){

    TNodo *fila;
    inicializeLI(&fila);

    if(!estaVaziaAVL(raiz)){

        insiraNoFinalLI(&fila, raiz);

        while(!estaVaziaLI(fila)){

                NodoAVL *nodo = removaDoInicio(&fila);

                printf(" %d |", nodo->info);
```

```
                if(nodo->esq != NULL){
                   insiraNoFinalLI(&fila, nodo->esq);
                }

                if(nodo->dir != NULL){
                   insiraNoFinalLI(&fila, nodo->dir);
                }
        }
    }
}
```

```
/* @brief       Função inserir_Nodo
*
*  @param       raiz - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*               valor - um número inteiro a ser inerido na árvore
*
*  @return      caso a inserção seja executada com sucesso:
*               um novo nodo criado contendo o valor passado como
paramentro.
*               caso falte memória ou já exista o valor na árvore:
*               o nodo passado como parâmetro retorna não havendo
modificação na árvore.
*/
NodoAVL *inserir_Nodo(NodoAVL *raiz, int valor){

        if(raiz == NULL){
```

```
            NodoAVL *aux;
            aux = (NodoAVL *)malloc(sizeof(NodoAVL));

            if(aux == NULL){
                  printf("Desculpe mas não foi possivel criar um novo nodo pois
falta memoria");
                  system("PAUSE");
                  return raiz;
            }

            aux->info = valor;
            aux->esq  = NULL;
            aux->dir  = NULL;

            return aux;
      }

      if(busca_Valor(raiz, valor)){
             printf("\nDesculpe mas a chave ja esta contida na arvore.\n\n");
             system("PAUSE");
             return raiz;
      }

      if(valor < raiz->info){

          raiz->esq = inserir_Nodo(raiz->esq, valor);
```

```
            if( altura( raiz->esq ) - altura( raiz->dir ) == 2 )
            {
               if( valor < raiz->esq->info )
                  raiz = rotateR( raiz );
               else
                  raiz = rotateLR( raiz );
               }


        }
        else if(valor > raiz->info){

           raiz->dir = inserir_Nodo(raiz->dir, valor);

            if( altura( raiz->dir ) - altura( raiz->esq ) == 2 )
            {
               if( valor > raiz->dir->info )
                  raiz = rotateL( raiz );
               else
                  raiz = rotateRL( raiz );
               }

        }

          return raiz;
}

/* @brief       Função percursos
```

```
*              Apresenta os percursos Em Ordem, Pre Ordem, Pos Ordem e Em Largura
*              referentes a árvore
*
*  @param       raiz - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      void
*/
void percursos(NodoAVL *raiz){



    printf("\n** Percurso em Pos Ordem ** \n[ ");
    posOrdem(raiz);
    printf("\b ]\n\n** Percurso Em Ordem **\n[ ");
    emOrdem(raiz);
    printf("\b ]\n\n** Percurso em Pre Ordem **\n[ ");
    preOrdem(raiz);
    printf("\b ]\n\n** Percurso Em Largura **\n[ ");
    emLargura(raiz);
    printf("\b ]\n\n");


}


/* @brief       Função estatisticas
*              Apresenta as estatisticas referentes a árvore
*
*  @param       raiz - um ponteiro (NodoAVL) para a raiz de uma arvore AVL
*
*  @return      void
```

```
*/
void estatisticas(NodoAVL *raiz){


    system("cls");
    printf("----------------------------\n");
    printf("|  Estatisticas da Arvore   |\n");
    printf("----------------------------\n");
    printf("\n** Altura da Arvore **\n\t %d\n", altura(raiz));
    printf("\n** Maior Valor **\n\t %d\n", maior_elemento(raiz));
    printf("\n** Menor Valor **\n\t %d\n", menor_elemento(raiz));
    printf("\n** Quantidade de elementos **\n\t %d\n",
num_elementos(raiz));
    percursos(raiz);
    system("PAUSE");


}


/*Fim da declaração das Funções manipulação da árvore*/


/
*****************************************************************************
*/


/* @brief       Função main
*             Função principal do programa
*
*  @param       argc - um inteiro
*             argv - um vetor de argumentos
*
*  @return      0 - caso o programa seja finalizado sem erros
```

```
*              1 - caso contrário
*/
int main(int argc, char *argv[])
{
    char op, read[2];
    int valor;
    NodoAVL *raiz;
    raiz = (NodoAVL *) malloc(sizeof(NodoAVL));
    raiz->esq = NULL;
    raiz->dir = NULL;
    raiz = NULL;

   do
    {
        system("cls");
        printf("\nEscolha uma opcao:\n\n");
        printf("I - Inserir\n");
        printf("E - Estatisticas\n");
        printf("S - Sair\n");
        printf("\nOpcao: ");
        scanf("%s", read);
        op = read[0];

        switch(op)
        {
                case 'I':
                case 'i':
                    system("cls");
```

```
                printf("\nDigite o valor a ser inserido na arvore.\n");
                printf("Valor:");
                scanf("%d", &valor);
                raiz = inserir_Nodo(raiz, valor);
                percursos(raiz);
                system("PAUSE");
                break;

            case 'E':
            case 'e':
                    estatisticas(raiz);
                    break;

            case 'S':
            case 's':
                destroi_arvore(&raiz);
                printf("\n\t** A arvore foi excluida com sucesso. **\n\t** O
programa sera finalizado em breve. **\n\n");
                system("PAUSE");
                exit(0);

            default:
                printf("\n  Opcao incorreta...\n\n\n");
                system("PAUSE");
                break;

        }
    }while(1);
```

```
}
```