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

typedef struct sInfo{
	int num;
}ELEM;

typedef struct sNo{
	ELEM info;
	struct sNo *esq;
	struct sNo *dir;
}NO;

void InserirRecursivo(NO **raiz, int elemento){ // INSERIR RECURSIVO
	if(*raiz==NULL){
		(*raiz)=criarNo();
		(*raiz)->esq=NULL;
		(*raiz)->dir=NULL;
		(*raiz)->info=elemento;
	}else
		if(elemento<(*raiz)->info){ //elemento menor que a raiz
			inserirRecursivo(&(*raiz)->esq,elemento);
		}
		if(elemento>(*raiz)->info){ // elemento maior que a raiz
			inserirRecursivo(&(*raiz)->dir,elemento);
		}
}

NO PesquisarElemento(NO **raiz, int elemento){ // PESQUISAR ELEMENTO
	if(raiz==NULL){
		return NULL;
	}
	if(raiz->info==elemento){
		return raiz;
	}
	else if(raiz->info>elemento){ // raiz maior que o elemento
		return pesquisarElemento(raiz->esq,elemento);
	}else
		return pesquisarElemento(raiz->dir,elemento);
}

void RemoverRecursivo(No **raiz,int elemento){ // REMOVER RECURSIVO
	NO *aux;

	if(vazia(raiz)){
		printf("ERRO");
		return;
	}
	if(elemento<(*raiz)->info){
		removerRecursivo(&(*raiz)->esq,elemento);
	}else
	if(elemento>(*raiz)->info){
		removerRecursivo(&(*raiz)->dir,elemento);
	}
	else
	aux=*raiz;
		if(((*raiz)->esq==NULL) && ((*raiz)->dir==NULL)){
			free(aux);
			(*raiz)=NULL;
		}
	else //aqui pra baixo
		if((*raiz)->esq==NULL){
			(*raiz)=(*raiz)->dir;
			aux->dir=NULL;
```

```
                free(aux);
        }else
                if((*raiz)->dir==NULL){
                        (*raiz)=(*raiz)->esq;
                        aux->esq=NULL;
                        free(aux);
        }
        else
                aux=maiorElemento(&(*raiz)->esq);
                aux->esq=(*raiz)->esq;
                aux->esq=(*raiz)->dir;
                free((*raiz));
                *raiz=aux;


}

NO *MaiorElemento(NO **no){ // REMOVER MAIOR ELEMENTO
        if((*no)->dir!=NULL){
                return maiorElemento(&(*no)->dir);
        }else
        NO *aux=*no;
        if((*no)->esq!=NULL){
                *no=(*no->esq);
        }else
                *no=NULL;
                return aux;

}

void PreOrdem (NO **raiz){ // PRE ORDEM
        if(vazia(raiz)){
                return ;
        }
        print("%d", (*raiz)->info);
        PreOrdem(&(*raiz)->esq);
        PreOrdem(&(*raiz)->dir);
}

void Ordem(NO **raiz){ // ORDEM
        if(vazia(raiz)){
                return ;
        }
        Ordem(&(*raiz)->esq);
        printf("%d", (*raiz)->info);
        Ordem(&(*raiz)->dir);
}

void PosOrdem(NO **raiz){ // POR ORDEM
        if(vazia(raiz)){
                return ;
        }
        Ordem(&(*raiz)->esq);
        Ordem(&(*raiz)->dir);
        printf("%d", (*raiz)->info);
}

void TrasPraFrente(NO **raiz){ // TRAS PRA FRENTE
        if(vazia(raiz)){
                return;
        }
        TrasPraFrente(&(*raiz)->dir);
        printf("%d", (*raiz)->info);
        TrasPraFrente(&(*raiz)->esq);
```

```
}

int AlturaArvore(NO **raiz){ // ALTURA
	if(vazia(raiz)){
		return 0;
	}
	int esquerda = 1+Altura(&(*raiz)->esq);
	int direita = 1+Altura(&(*raiz)->dir);
	if(esquerda>direita){
		return esquerda;
	}
	return direita;
}

int QntNos(NO **raiz){ // QUANTIDADE DE NOS OU QUANTIDADE DE SUB-ARVORES
	if(vazia(raiz)){
		return 0;
	}
	return 1 + QntNos(&(*raiz)->esq) + QntNos(&(*raiz)->dir);
}

void PrintFolhaEmOrdem(NO **raiz){ // PRINT FOLHA EM ORDEM ( pode mudar a ordem
do print para primeiro e depois chamar as funcao esq e dir)
	if(vazia(raiz)){
		return ;
	}
	PrintFolhaEmOrdem(&(*raiz)->esq);
	if((*raiz)->esq==NULL && (*raiz)->dir==NULL){
		print("%d", (*raiz)->info);
	}
	PrintFolhaEmOrdem(&(*raiz)->dir);
}

int InserirIterativo(NO **raiz, ALUNO i){ // INSERIR ITERATIVO
	NO *novo = criarno();
	if(novo == NULL)
		return 0;
	novo->info = i;
	novo->dir = NULL;
	novo->esq = NULL;

	if(vazia(raiz)){
		*raiz = novo;
		return 0;
	}
	NO *aux = *raiz;
	NO *ant = NULL;

	while(aux!=NULL){
		ant = aux;
		if(i.matricula >= aux->info.matricula)
			aux = aux->dir;
		else
			aux = aux->esq;
	}
	if(i.matricula > aux->info.matricula)
		ant->dir = novo;
	else
		ant->esq = novo;
	return 1;
}

int AlturaNo(NO **raiz,int n){ // ALTURA DE UM NO DIGITADO PELO USUARIO
	if (vazia(raiz)){
```

```
            return 1;
      }
      if (n==(*raiz)->info){
            return 1;
      }
      if(n>(*raiz)->info){
            return 1 + AlturaNo(&(*raiz)->dir,n)
      }
      return 1 + AlturaNo(&(*raiz)->esq,n)
}

int NivelNo(NO **raiz,int n){ // NIVEL DE UM NO DIGITADO PELO USUARIO
      if (vazia(raiz)){
            return 1;
      }
      if (n==(*raiz)->info){
            return 0;
      }
      if(n>(*raiz)->info){
            return 1 + NivelNo(&(*raiz)->dir,n)
      }
      return 1 + NivelNo(&(*raiz)->esq,n)
}

void ExcluirArvore(NO **raiz){ //  EXCLUIR ARVORE
      if(vazia(raiz)){
            return;
      }
      ExcluirArvore(&(*raiz)->esq);
      ExcluirArvore(&(*raiz)->dir);
      free((*raiz));
      (*raiz)=NULL;
}

int Grau(NO **raiz){ // RETORNAR O GRAU DA ARVORE
      if((*raiz)->esq==NULL && (*raiz)->dir==NULL){
            return 0;
      }
      if((*raiz)->esq==NULL || (*raiz)->dir==NULL){
            return 1
      }
      return 2;
}

int Grau(NO **raiz,int elem){ // RETORNAR O GRAU DE UM ELEMENTO DIGITADO E
VERIFICANDO
      NO *pesq=Pesquisar(raiz,elem);
      if(pesq==NULL){
            return -1;
      }

      if((*raiz)->esq==NULL && (*raiz)->dir==NULL){
            return 0;
      }
      if((*raiz)->esq==NULL || (*raiz)->dir==NULL){
            return 1
      }
      return 2;
}

int QntFolha(NO **raiz){  // QUANTIDADE DE FOLHA
      if(vazia(raiz)){
            return 0;
      }
```

```
	int cont=0;
	if((*raiz)->esq==NULL && (*raiz)->dir==NULL){
		cont++;
	}
	return cont + QntFolha(&(*raiz)->esq)+QntFolha(&(*raiz)->dir);
}

int Soma(NO **raiz){ // SOMA TODOS OS ELEMENTOS DO ARVORE
	if(vazia(raiz)){
		return 0;
	}
		return (*raiz)->info + Soma(&(*raiz)->esq) + Soma(&(*raiz)->dir);
	return 0;
}

int Nivel(NO **raiz){ // NIVEL DA ARVORE
	if(vazia(raiz)){
		return 1;
	}
	if((*raiz)->esq==NULL && (*raiz)->dir==NULL){
		return 0;
	}
	int esquerda= 1 + Nivel((*raiz)->esq);
	int direita= 1 + Nivel((*raiz)->dir);
	if(esquerda>direita){
		return esquerda;
	}
	return direita;
}
```