```c
#include <stdio.h>
#include <conio.h>
#include <malloc.h>

typedef int telem;
typedef struct no{
        struct no* esq;
        telem info;
        struct no* dir;
} tno;
typedef tno* tarvbin;

/*1) Criar uma árvore vazia*/

void criar(tarvbin *T){
     *T = NULL;
}

/*2) Verifica se árvore vazia ou não
Retorna 1 se árvore estiver vazia, 0 caso contrário.*/

int vazia(tarvbin T){
    return (T == NULL);
}

/*3) Buscar um elemento na árvore
Busca um elemento na árvore, retornando o seu endereço, caso o encontre.
Se o elemento não for encontrado, retorna o endereço nulo (NULL).*/

tarvbin busca(tarvbin T, telem dado){
        tarvbin achou;
        if (T == NULL) return NULL;
        if (T->info == dado) return T;
        achou = busca(T->esq, dado);
        if (achou == NULL) achou = busca(T->dir, dado);
        return achou;
}

/*4) Inserir um nó raiz
Insere um nó raiz numa árvore vazia. Retorna 1 se a inserção for bem
sucedida, ou 0 caso contrário.*/

int ins_raiz(tarvbin *T, telem dado){
    tarvbin novo;
    if (*T != NULL) return 0; /* erro: já existe raiz */
    novo = (tno*) malloc(sizeof(tno));
    if (novo == NULL) return 0; /* erro: memória insuficiente */
    novo->info = dado;
    novo->esq = novo->dir = NULL;
    *T = novo;
    return 1;
}

/*5) Inserir um filho à direita de um dado nó*/

int ins_dir(tarvbin T, telem pai, telem filho){
    tarvbin f, p, novo;
    /* verifica se o elemento já não existe */
    f = busca(T,filho);
    if (f != NULL) return 0; /* erro: dado já existente */
    /* busca o endereço do pai e verifica se já não possui filho direito */
```

```
    p = busca(T,pai);
    if (p == NULL) return 0; /* erro: pai não encontrado */
    if (p->dir != NULL) return 0; /* erro: já existe filho direito */
    novo = (tno*) malloc(sizeof(tno));
    if (novo == NULL) return 0; /* erro: memória insuficiente */
    novo->info = filho;
    novo->esq = novo->dir = NULL;
    p->dir = novo;
    return 1;
}

int ins_esq(tarvbin T, telem pai, telem filho){
    tarvbin f, p, novo;
    /* verifica se o elemento já não existe */
    f = busca(T,filho);
    if (f != NULL) return 0; /* erro: dado já existente */
    /* busca o endereço do pai e verifica se já não possui filho esquerdo */
    p = busca(T,pai);
    if (p == NULL) return 0; /* erro: pai não encontrado */
    if (p->esq != NULL) return 0; /* erro: já existe filho esquerdo */
    novo = (tno*) malloc(sizeof(tno));
    if (novo == NULL) return 0; /* erro: memória insuficiente */
    novo->info = filho;
    novo->esq = novo->dir = NULL;
    p->esq = novo;
    return 1;
}

/*6) Esvaziar uma árvore
Desaloca todo o espaço de memória da árvore e retorna a árvore ao estado
equivalente ao define, isto é, nula.
Utiliza o algoritmo de Pós-Ordem para percurso.*/

void esvaziar(tarvbin *T){
     if (*T == NULL) return;
     esvaziar(&(*T)->esq);
     esvaziar(&(*T)->dir);
     free(*T);
     *T = NULL;
}

/*7) Exibir a árvore
Um procedimento recursivo para exibir a árvore, usando um percurso pré-ordem,
poderia ser o seguinte:

void exibir(tarvbin T, int col, int lin, int desloc){
     // col e lin são as coordenadas da tela onde a árvore irá iniciar,
     // ou seja, a posição da raiz, e desloc representa o deslocamento na
tela
     // (em colunas) de um nó em relação ao nó anterior.

     if (T == NULL) return; // condição de parada do procedimento recursivo
     gotoxy(col,lin);
     printf("%d",T->info);
     if (T->esq != NULL)
        exibir(T->esq,col-desloc,lin+2,desloc/2+1);
     if (T->dir != NULL)
        exibir(T->dir,col+desloc,lin+2,desloc/2+1);
}          */

void exibir(tarvbin T){
```

```
     if (T == NULL) return; // condição de parada do procedimento recursivo

     printf("\n%d\n",T->info);
     if (T->esq != NULL)
        exibir(T->esq);
     if (T->dir != NULL)
        exibir(T->dir);
}

int main(int argc, char *argv[])
{
  tarvbin A;

  criar (&A);

  ins_raiz(&A, 5);

  ins_dir(A, 5, 6);

  ins_esq(A, 5, 4);

  exibir(A);

  getchar();

  return 0;
}
```