```c
/*Esercizio 1. Si definisca una funzione C che,
ricevendo in input un albero binario di interi t,
restituisce il livello della foglia piu' a sinistra.*/

/*Soluzione 1*/
int livFoSin(TreePtr t)
/*liv deve essere inizializzato a -1
*postc: se t e' vuoto restituisce -1, altrimenti
liv contiene il livello della foglia più a sinistra  di tPtr */
{if (!t) return -1;
if (t -> lPtr) return livFoSin(t -> lPtr) +1;
else return livFoSin(t -> rPtr) +1;}

/*Soluzione 2*/

void livFoSin2(TreePtr tPtr, int* liv)
/*liv deve essere inizializzato a -1
*postc: se tPtr e' vuoto restituisce -1, altrimenti
liv contiene il livello della foglia più a sinistra  di tPtr */
{if (!tPtr) return;
if (tPtr -> lPtr) {(*liv)=(*liv)+1;livFoSin2(tPtr -> lPtr,liv);}
else {(*liv)=(*liv)+1; livFoSin2(tPtr -> rPtr,liv);}
 }


/*Esercizio 2. Si definisca una funzione C che,
ricevendo in input un albero binario di interi t e un un intero k,
restituisce il numero delle foglie di t a livello k.*/

/*Soluzione 1*/
int  foglieLivK(TreePtr tPtr, int k)
/*restituisce il numero di foglie di livello  k.
* prec: k>=0;
* postc: restituisce 0 se l'albero e' vuoto, altrimenti il numero
delle foglie di livello  k */
{if (!tPtr ) return 0;
if (foglia(tPtr) && k==0 )  return 1;
return foglieLivK(tPtr -> lPtr,k-1) + foglieLivK(tPtr -> rPtr,k-1) ;
}

/*Soluzione 2*/
void  foglieLivK2(TreePtr tPtr,int* numF, int k)
/*numF deve essere inizializzato a 0.
* prec: k>=0;
* postc: calcola in numF  il numero di foglie la cui distanza dalla radice e' k,
se l'albero e' vuoto numF rimane a  0*/
{if (!tPtr ) return ;
if (foglia(tPtr) && k==0 ) {(*numF)++; return ;}
foglieLivK2(tPtr -> lPtr,numF,k-1);
foglieLivK2(tPtr -> rPtr,numF,k-1) ;
}

/*Esercizio 3. Si definisca una funzione C che,
```

```
ricevendo in input un albero binario di interi t,
restituisce la lunghezza del cammino interno,
cioe' la somma dei livelli dei nodi interni.*/


/*Soluzione 1*/
int cammInt(TreePtr tPtr)
/* Calcola la lunghezza del cammino interno, cioè
 la somma dei livelli di tutti i nodi interni.
 postc:restituisce la lunghezza del cammino interno, -1 se l'albero è vuoto */
{int liv = 0;
if (tPtr ) return cammIntAus(tPtr,liv);
else return -1;}

int cammIntAus(TreePtr tPtr, int liv)
 /* prec: tPtr != NULL
* postc:   restituisce la lunghezza del cammino interno di un albero  non nullo
* calcola in liv il livello di tPtr  */
{int cammD,cammS;
if (!tPtr -> left  && !tPtr -> right  ) return 0;
if (tPtr -> left) cammS = cammIntAus(tPtr -> left,liv+1) ; else cammS = 0;
if (tPtr -> right)  cammD = cammIntAus(tPtr -> right,liv+1); else cammD = 0;
return cammS+cammD+liv;}

/*Soluzione 2*/
int cammInt(TreePtr tPtr)
/* postc:restituisce la lunghezza del cammino interno, -1 se l'albero è vuoto */
{int ris =0,liv = 0;
if (tPtr )
{ CammIntAus(tPtr,liv,&ris);return ris;}
 else return -1;}

void CammIntAus(TreePtr tPtr, int liv,int * ris)
 /* prec: tPtr != NULL
* postc:   calcola in ris la lunghezza del cammino interno di un albero  non nullo
* e in liv il livello del nodo corrente   */
{if (foglia(tPtr)) return;
if (tPtr -> left ) CammIntAus(tPtr -> left,liv+1,ris);
if (tPtr -> right) CammIntAus(tPtr -> right, liv+1,ris);
(*ris)+=liv;
}
```