Esercizio n.1. (sbarramento)

Si definisca una funzione ricorsiva C che restituisce la fusione di due liste concatenate ordinate in una lista concatenata ordinata che contiene gli elementi di entrambe. Non si usi memoria aggiuntiva. Il prototipo della funzione è

ListaP merge(ListaP L, ListaP M);

dove ListaP è il nome del tipo puntatore a Lista, la struttura ricorsiva per gli elementi della lista concatenata.

Esercizio n. 2

Si definisca una funzione C che restituisce il numero di nodi hs-sbilanciati nell'albero in input, -1 se quest'ultimo è vuoto. Un nodo è hs-sbilanciato se il suo sottoalbero sinistro è più alto di quello destro. La funzione ha il seguente prototipo

int **sbilHS**(TreePtr tPtr);

Esercizio n. 3

Si fornisca la specificazione formale in C della classe polinomio, la cui descrizione è sotto riportata.

Specificazione informale della classe polinomio
Un polinomio di grado *d* , $p(x) = a_0 + a_1x + a_2x^2 + ... + a_dx^d$, può essere rappresentato in memoria semplicemente memorizzando i suoi coefficienti a_k, per $0 \leq k \leq d$, in un vettore. Si vuole creare una classe per valutare e sommare polinomi.

Requisiti
Quando si crea un polinomio i coefficienti sono posti a 0.
Si assume che i coefficienti di x^k di un polinomio di grado d, per $k>d$, valgono 0.

Dato un polinomio si deve poter ottenere il suo grado e un suo coeffiente.
Si deve poter stampare a video il grado e i coefficienti di un polinomio.
Si vuole poter valutare un polinomio e sommare due polinomi.
Per sommare due polinomi si deve creare un nuovo polinomio con grado e coefficienti opportuni.

Sol. Es 1.

```
ListaPtr merge(ListaPtr L1,ListaPtr L2)
/*fonde due liste concatenate ordinate in una
Prec: L1 e L2 sono ordinate
Post: restituisce la lista ordinata fusione delle liste in input*/
{if (!L1) return L2;
if (!L2) return L1;
if (L1 -> elem < L2 -> elem)
{ L1 -> next = merge(L1 -> next,L2);
return L1;}
else
{L2 -> next =merge(L1,L2->next);
return L2;}
}
```

Sol. Es 2.

```
int sbilS(TreePtr T)
/*Post: restituisce  il numero dei nodi hs-sbilanciati in  T, cioe' quei nodi
* radici di sottoalberi di T il cui sottoalbero sinistro e' piu' alto di quello
destro, -1 se l'albero in input è vuoto */
{int ris = 0;
if (!T)  return -1;
sbilSAus(T,&ris);
return ris;}

int sbilSAus(TreePtr T, int* ris)
/*La funzione deve essere chiamata con *ris =0.
Post: restituisce l'altezza di T e mette in ris il numero dei nodi hs-sbilanciati
in  T */
{int hl,hr;
if (!T)  return -1;
```

```
hl = sbilSAus(T->left,ris) ;
hr = sbilSAus(T->right,ris);
if (hl  >  hr) {(*ris)++; return hl+1;}  /*il sinistro e' il sottoalbero piu' alto*/
else  return hr +1;} /*e' il destro a essere della stessa altezza o piu' alto*/
```

Sol. Es. 3

```
/** Poli.h  specificazione classe polinomio

**/
typedef struct polinomio * Poli;

/* Costruttore */
Poli costPoli( int n );
/* prec: n>0
postc: construisce un polinomio di grado n, a coifficienti nulli */

/* distruttore */
void distrPoli( Poli p );
/* libera la memoria impegnata da p,
prec: p!= NULL*/


int grado( Poli p );
/* prec: p!= NULL
postc:restituisce il grado del polinomio p*/

int Coeff( Poli p, int j );
 /* prec: p!= NULL && j>=0
postc:restituisce il j-simo coeffieciente del polinomio p, 0<=j<=grado(p), 0
altrimenti*/


int insValCoeff( Poli p, int j, int x );
 /* prec: p!= NULL && j>=0
postc:pone a x il j-simo coeffieciente del polinomio p,
 restituisce 1 se 0<=j<=grado del polinomio, 0 altrimenti*/

int Eval( Poli p, int x );
 /* prec: p!= NULL
postc:restituisce il valore del polinomio in x*/
```

```
void stampaPoli( Poli p );
/*stampa a video il polinomio*/


Poli somma( Poli a, Poli b );
 /*  prec: a!= NULL && b != NULL
  postc:restituisce il polinomio somma di a e b*/
```