Laboratorio Informatica A

Problema 1

Partendo dal file alberi.c fornito, scrivere la funzione che calcola la somma di tutti i nodi dell'albero contenenti un valore pari.

Problema 2

Partendo dal file alberi.c fornito, scrivere la funzione che calcola il prodotto di tutti i nodi dell'albero contenenti un valore divisibile per 3.

Problema 3

Partendo dal file alberi.c fornito, scrivere la funzione massimoPari che restituisce il massimo elemento pari dell'albero.

(per comodita' il file e' cmq riportato in coda al doc. corrente)

Problema 4

Scrivere una funzione (e opportuna funzione main per testarla) che dato un albero in input, inserisca in una lista i soli nodi di livello pari.

Problema 5

Modificate la struttura dati dell'albero definito nel file alberi.c in modo da poter salvare in ciascun nodo un intero che rappresenta la profondità del nodo stesso all'interno dell'albero. Ad esempio il nodo radice avrà profondità 0, i suoi figli profondità 1, i figli dei figli profondità 2 e così via. Scrivere una funzione che esplora tutto l'albero, calcola la profondità di ogni nodo e salva tale informazione all'interno del campo appena aggiunto alla struttura dell'albero.

Problema 6

Si consideri la seguente definizione di un albero binario:

```
typedef struct ET { char dato;
                    struct ET * left, * right; } treeNode;
typedef treeNode * tree;
```

Si codifichi in C la seguente funzione:

```
int contains (tree t, char word[])
```

che restituisce 1 se concatenando le lettere trovate percorrendo uno dei cammini dalle foglie alla radice nell'albero si ottiene la parola contenuta nell'array word, 0 altrimenti. Si scriva anche un'opportuna funzione main per testare contains.

Problema 7

Un albero binario di ricerca è un albero binario in cui i valori dei figli di un nodo sono ordinati in modo da avere i valori minori di quelli del nodo di partenza nei figli a sinistra e valori più grandi nei figli a destra. Tale proprietà è valida per ogni nodo dell'albero. Dichiarate e definite poi una funzione insertInTree che, data una lista d'interi li inserisce uno a uno in un albero inizialmente vuoto e costruisce un albero binario di ricerca.

Problema 8

Partendo dalla soluzione del precedente esercizio, dichiarate e definite una funzione insertInTreeBalanced che inserisce i numeri interi uno a uno in un albero inizialmente vuoto e che costruisce un albero perfettamente bilanciato (un albero bilanciato è un albero in cui tutte le foglie hanno la stessa profondità a meno di un passo). Per costruire un albero perfettamente bilanciato non dovete inserire i numeri nell'albero così come sono presenti nella lista, ma dovete prima ordinare la lista (si suggerisce di definire una funzione per ordinare la lista) e poi navigarla dividendo ricorsivamente in due parti uguali la lista. Ad esempio, data la lista iniziale {5,7,2,1,8,9,3}, prima ordinate la lista in modo da ottenere {1,2,3,5,7,8,9}, poi inserite gli elementi nell'albero nel seguente ordine 5-2-8-1-3-7-9. In pratica la funzione procede ricorsivamente dividendo in due la lista iniziale, inserendo l'elemento mediano (i.e., quello che separa le due sotto-liste), e procede poi con lo stesso algoritmo per le due sotto-liste identificate.

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

typedef struct s_nodo {
	int val;
	struct s_nodo *left;
	struct s_nodo *right;
} nodo;
typedef nodo *albero;



albero creaAlbero();
albero createVal(int val);

void print(albero t);

int f(albero t);

int main(){
  int ris=0;
  albero alb = creaAlbero();
  print(alb);

  //chiamata alla funzione da scrivere
  ris=f(alb);

  printf("\n\n%d\n\n", ris);
  return 0;
}

int f(albero t) {
    int ris=0;
    //funzione da scrivere

    return ris;
}

albero creaAlbero() {
	albero tmp = createVal(7);
	tmp->left = createVal(3);
	tmp->left->left = createVal(9);
	tmp->left->right = createVal(10);
	tmp->right = createVal(8);
	tmp->right->left = createVal(5);
	tmp->right->right = createVal(12);
	tmp->right->right->left = createVal(11);
	tmp->right->right->right = createVal(6);

	return tmp;
}

albero createVal(int val) {
	albero tmp = malloc(sizeof(nodo));
	tmp->val = val;
	tmp->left = NULL;
	tmp->right = NULL;
	return tmp;
}

void print(albero t){
```

```
    if(t==NULL)
       return;
    printf(" (");
    print(t->left);
    printf(" %d ",t->val);
    print(t->right);
    printf(") ");
}
```