KIT COMPLETO - Programmazione 2 (Liste, Alberi, Esercizi)

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

/* === LISTE ITERATIVE === */

typedef struct node {
    int data;
    struct node *next;
} IntNode, *IntList;

void stampa(IntList l) {
    while (l != NULL) {
        printf("%d ", l->data);
        l = l->next;
    }
    printf("\n");
}

int length(IntList l) {
    int len = 0;
    while (l != NULL) {
        len++;
        l = l->next;
    }
    return len;
}

int somma(IntList l) {
    int somma = 0;
    while (l != NULL) {
        somma += l->data;
        l = l->next;
    }
    return somma;
}

IntList inserisciTesta(IntList l, int val) {
    IntList nuovo = malloc(sizeof(IntNode));
    nuovo->data = val;
    nuovo->next = l;
    return nuovo;
}

IntList inserisciCoda(IntList l, int val) {
    IntList nuovo = malloc(sizeof(IntNode));
    nuovo->data = val;
    nuovo->next = NULL;
    if (l == NULL) return nuovo;
```

```
    IntList tmp = l;
    while (tmp->next != NULL) tmp = tmp->next;
    tmp->next = nuovo;
    return l;
}

IntList rimuovi(IntList l, int val) {
    IntList curr = l, prev = NULL;
    while (curr != NULL) {
        if (curr->data == val) {
            if (prev == NULL) l = curr->next;
            else prev->next = curr->next;
            free(curr);
            return l;
        }
        prev = curr;
        curr = curr->next;
    }
    return l;
}

void libera(IntList l) {
    while (l != NULL) {
        IntList tmp = l;
        l = l->next;
        free(tmp);
    }
}

/* === LISTE RICORSIVE === */

void stampaRic(IntList l) {
    if (l != NULL) {
        printf("%d ", l->data);
        stampaRic(l->next);
    }
}

int lengthRic(IntList l) {
    if (l == NULL) return 0;
    return 1 + lengthRic(l->next);
}

int sommaRic(IntList l) {
    if (l == NULL) return 0;
    return l->data + sommaRic(l->next);
}

IntList inserisciCodaRic(IntList l, int val) {
    if (l == NULL) {
        IntList nuovo = malloc(sizeof(IntNode));
        nuovo->data = val;
        nuovo->next = NULL;
        return nuovo;
```

```
    }
    l->next = inserisciCodaRic(l->next, val);
    return l;
}

IntList rimuoviRic(IntList l, int val) {
    if (l == NULL) return NULL;
    if (l->data == val) {
        IntList tmp = l->next;
        free(l);
        return tmp;
    }
    l->next = rimuoviRic(l->next, val);
    return l;
}

void liberaRic(IntList l) {
    if (l != NULL) {
        liberaRic(l->next);
        free(l);
    }
}

/* === ALBERI BINARI === */

typedef struct treeNode {
    struct treeNode *left;
    int data;
    struct treeNode *right;
} IntTreeNode, *IntTree;

void stampaInOrder(IntTree t) {
    if (t != NULL) {
        stampaInOrder(t->left);
        printf("%d ", t->data);
        stampaInOrder(t->right);
    }
}

int contaNodi(IntTree t) {
    if (t == NULL) return 0;
    return 1 + contaNodi(t->left) + contaNodi(t->right);
}

int sommaNodi(IntTree t) {
    if (t == NULL) return 0;
    return t->data + sommaNodi(t->left) + sommaNodi(t->right);
}

int altezza(IntTree t) {
    if (t == NULL) return 0;
    int sx = altezza(t->left);
    int dx = altezza(t->right);
    return 1 + (sx > dx ? sx : dx);
```

```
}

void mirror(IntTree t) {
    if (t == NULL) return;
    IntTree tmp = t->left;
    t->left = t->right;
    t->right = tmp;
    mirror(t->left);
    mirror(t->right);
}

void liberaAlbero(IntTree t) {
    if (t != NULL) {
        liberaAlbero(t->left);
        liberaAlbero(t->right);
        free(t);
    }
}

/* === ESERCIZI STILE ESAME === */

// Es. 1 - Check stringhe ricorsivo
_Bool check(const char *s1, int n1, const char *s2, int n2) {
    if (n2 == 0) return 1;
    if (n1 != 2 * n2) return 0;
    if (*s1 != *s2) return 0;
    return check(s1 + 2, n1 - 2, s2 + 1, n2 - 1);
}

// Es. 2 - Conta nodi tra profondità
typedef struct treeNodeChar {
    struct treeNodeChar *left;
    char data;
    struct treeNodeChar *right;
} CharTreeNode, *CharTree;

int count_helper(CharTree t, int d, int m, int n) {
    if (t == NULL) return 0;
    int cont = (d >= m && d <= n) ? 1 : 0;
     return cont + count_helper(t->left, d + 1, m, n) + count_helper(t->right, d + 1, m,
n);
}

int count(CharTree tree, int m, int n) {
    return count_helper(tree, 0, m, n);
}

// Es. 3 - Transfer nodi comuni
int contiene(IntList l, int val) {
    while (l != NULL) {
        if (l->data == val) return 1;
        l = l->next;
    }
    return 0;
```

```
}

IntList transfer(IntList *lsPtr1, IntList ls2) {
    IntList newHead = NULL, newTail = NULL;
    IntList prev = NULL, curr = *lsPtr1;

    while (ls2 != NULL) {
        prev = NULL;
        curr = *lsPtr1;
        while (curr != NULL) {
            if (curr->data == ls2->data) {
                IntList toMove = curr;
                if (prev == NULL)
                    *lsPtr1 = curr->next;
                else
                    prev->next = curr->next;

                curr = curr->next;
                toMove->next = NULL;

                if (newHead == NULL)
                    newHead = newTail = toMove;
                else {
                    newTail->next = toMove;
                    newTail = toMove;
                }
                break;
            } else {
                prev = curr;
                curr = curr->next;
            }
        }
        ls2 = ls2->next;
    }

    return newHead;
}
```