```cpp
#include <iostream>
#include <fstream>
using namespace std;

enum Materiale {VETRO, LEGNO};

struct sfera{
    Materiale mat;
    sfera *pun;
};

struct ListaSfere{
    sfera *testa;
};

void inizializzaListaSfere(ListaSfere &LS){
    LS.testa = NULL;
}


/* la funzione inserisciSfera non può inserire nel mezzo della
 * lista ma può solo inserire in testa, se la lista è vuota
 * o se il materiale m è diverso dal materiale del primo
 * elemento, oppure può inserire in fondo se il materiale
 * dell'ultimo elemento è diverso da m.
 * In tutti gli altri casi,l'inserimento fallisce.
 * ESEMPIO:
 * ho la lista di sfere <Vetro, legno, vetro, legno>
 * - se voglio inserire una sfera di vetro, me la inserisce in
 *   fondo, altrimenti avrei due sfere consecutive
 *   dello stesso materiale vicino
 * - se voglio inserire una sfera di legno, la posso inserire
 *   solo in testa
 */
bool inserisciSfera(ListaSfere &LS, Materiale m){
    sfera *q;

    // Caso inserimento in testa
    if((LS.testa==NULL) || (LS.testa->mat != m)) {
        q = new sfera;
        q->mat = m;
        q->pun = LS.testa;
        LS.testa = q;
        return true;
    }

    // Non posso inserire in testa, scorro per un'eventuale
    // inserimento in fondo

    for(q=LS.testa;q->pun!=NULL; q=q->pun);
    //alla fine q punta all'ULTIMA sfera

    if(q->mat != m){
        //Caso favorevole, posso inserire
        q->pun = new sfera;
        q = q->pun;
        q->mat = m;
```

```
        q->pun = NULL;
        return true;
    }

    //Caso sfavorevole, inserimento fallito,
    //ritorno false senza fare modifiche
    return false;
}

/* Anche la funzione elimina segue lo stesso ragionamento
della
 * inserisci: posso estrarre o il primo elemento se ha
 * materiale m, oppure posso estrarre l'ultimo elemento se ha
 * materiale m. In tutti gli altri casi (nel mezzo della lista
)
 * non posso eliminare nulla.
 * Attenzione alle situazioni di errore.
 */
bool eliminaSfera(ListaSfere &LS, Materiale m){
    if(LS.testa == NULL)
        return false;

    sfera *p, *q;
    //Controllo se il primo elemento ha materiale m,
    // ed eventualmente lo elimino
    if(LS.testa->mat == m){
        q = LS.testa;
        LS.testa = LS.testa->pun;
        delete q;
        return true;
    }

    //Non ho potuto estrarre in testa,
    // provo in fondo alla lista
    for(q=LS.testa;q->pun!=NULL;q=q->pun)
        p=q;

    //Alla fine del for q punta all'ultimo elemento
    //p al penultimo
    if(q->mat == m){
        p->pun = NULL;
        delete q;
        return true;
    }

    //Caso sfortunato, non posso estrarre senza
    //rispettare il vincolo, ritorno false
    return  false;
}


void stampaListaSfere(ListaSfere LS){
    cout << '<';
    sfera *q = LS.testa;
    while(q!=NULL){
        if(q->mat == VETRO)
            cout << "VETRO";
```

```
            else
                cout << "LEGNO";
            if(q->pun!=NULL)
                cout << ", ";
            q = q->pun;
        }
        cout << '>' << endl;
}

void stampaListaSfere(ListaSfere LS, const char *str){
        ofstream out(str);
        if(!out)
            return;

        out << '<';
        sfera *q = LS.testa;
        while(q!=NULL){
            if(q->mat == VETRO)
                out << "VETRO";
            else
                out << "LEGNO";
            if(q->pun!=NULL)
                out << ", ";
            q = q->pun;
        }
        out << '>' << endl;
}

void cambiaMateriali(ListaSfere &LS){
        sfera *q = LS.testa;
        while(q!=NULL){
            if(q->mat == VETRO)
                q->mat = LEGNO;
            else
                q->mat = VETRO;
            q = q->pun;
        }
}

ListaSfere sottoListaSfere(ListaSfere LS, int k){

        ListaSfere nuova;
        inizializzaListaSfere(nuova);

        if(k<=0 || LS.testa == NULL)
            return nuova;

        nuova.testa = new sfera;
        nuova.testa->mat = LS.testa->mat;
        nuova.testa->pun = NULL;
        int conta = 1;

        //CON p scorriamo la lista in cui copiare,
        // con q la lista da cui copiare
        sfera *p = nuova.testa;
        for(sfera *q = LS.testa->pun; q!=NULL && conta<k; q=q->pun
){
```

```
            p->pun = new sfera;
            p = p->pun;
            p->mat = q->mat;
            p->pun = NULL;
            conta++;
        }

        return nuova;
    }

    void eliminaListaSfere(ListaSfere &LS){
        sfera *q = LS.testa;
        while(LS.testa != NULL){
            LS.testa = LS.testa->pun;
            delete q;
            q = LS.testa;
        }
    }

    int main() {
        ListaSfere LS;
        inizializzaListaSfere(LS);
        stampaListaSfere(LS);        // <>

        inserisciSfera(LS, VETRO);   //inserisce
        inserisciSfera(LS, LEGNO);   //inserisce
        inserisciSfera(LS, LEGNO);   //inserisce
        inserisciSfera(LS, LEGNO);   //non inserisce
        inserisciSfera(LS, VETRO);   //inserisce
        stampaListaSfere(LS);        // <VETRO, LEGNO, VETRO, LEGNO
    >

        eliminaSfera(LS, LEGNO);
        stampaListaSfere(LS);        // <VETRO, LEGNO, VETRO>

        eliminaSfera(LS, LEGNO);     // Eliminazione fallita
        stampaListaSfere(LS);        // <VETRO, LEGNO, VETRO>

        eliminaSfera(LS, VETRO);
        stampaListaSfere(LS);        // <LEGNO, VETRO>

        cambiaMateriali(LS);
        stampaListaSfere(LS);        // <VETRO, LEGNO>

        ListaSfere LS1 = sottoListaSfere(LS, 1);
        cout << "Sottolista: ";
        stampaListaSfere(LS1);       // Sottolista: <VETRO>

        eliminaListaSfere(LS);
        stampaListaSfere(LS);        // <>

        stampaListaSfere(LS1, "output.txt");
        //Nel file stampa <VETRO>

        return 0;

    }
```