```cpp
#ifndef LIST_H
#define LIST_H
#include<iostream>
using std::cerr;

struct Node
{
    Node(int x) { data = x; nxt = nullptr; }
    Node* nxt = nullptr;
    int data;
};

enum InsertionWay { IW_BEFORE, IW_AFTER };
enum RemoveWay { RW_EXACT, RW_BEFORE, RW_AFTER };

class List
{
public:
    ~List() {
        Node* p, * q;
        p = head;
        for (; p;) { q = p; p = p->nxt; delete q; }
    }
    List():cnt(0) { head = new Node(-1);head->nxt = nullptr; };
    int getcnt()const { return cnt; }
    void insert(int data, int whereData, InsertionWay way = IW_AFTER);
    void remove(int whereData, RemoveWay way = RW_EXACT);
    void setHead(int data) { head->nxt = new Node(data); cnt = 1; }
    Node* getHead()const { return head; }

private:

    bool __insertionAfter(int data, Node* prev)
    {
        if (prev == nullptr) { cerr << "Empty Node!"; exit(1); }
        if (prev->nxt == nullptr) { prev->nxt = new Node(data); return 0; }
        Node* newNode = new Node(data);
        newNode->nxt = prev->nxt;
        prev->nxt = newNode;
        cnt++;
        return 1;
    }

    bool __deleteAfter(Node* prev)
    {
        if (prev == nullptr) { /*cerr << "Empty Node!"; exit(1);*/return 0; }
        if (prev->nxt == nullptr) { cerr << "Empty Node!"; exit(1); }
        Node* deleteNode = prev->nxt;
        prev->nxt = prev->nxt->nxt;
        delete deleteNode;
        cnt--;
        return 1;
```

```
    }

    Node* __findFather(int data);
    Node* __findGrandFather(int data);


    Node* head;
    int cnt;

};

void List::insert(int data, int whereData, InsertionWay way)
{
    if (way == InsertionWay::IW_AFTER)  { __insertionAfter(data,
__findFather(whereData)->nxt); }
    if (way == InsertionWay::IW_BEFORE) { __insertionAfter(data,
__findFather(whereData));      }
}

void List::remove(int whereData, RemoveWay way)
{
    if (way == RemoveWay::RW_EXACT)     {
__deleteAfter(__findFather(whereData));               }
    if (way == RemoveWay::RW_BEFORE)    {
__deleteAfter(__findGrandFather(whereData));          }
    if (way == RemoveWay::RW_AFTER)     { __deleteAfter(__findFather(whereData)-
>nxt);          }

}

Node* List::__findFather(int data)
{
    Node* p = head;
    for (; (p->nxt) && p->nxt->data != data; p = p->nxt) { }
    if (p->nxt) return p;
    else return nullptr;
}

Node* List::__findGrandFather(int data)
{
    Node* p = head;
    for (; p->nxt && p->nxt->nxt && p->nxt->nxt->data != data; p = p->nxt) { }
    if (p->nxt && p->nxt->nxt) return p;
    else return nullptr;

    return p;
}


#endif
```