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

/******* CODE FOR LINKED LIST ********/

typedef struct Node{
    float x;
    struct Node *next; // The next node in the list
}Node;

// Get the address of a newly created structure variable
Node* createNode(float x){
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->x = x; // We know the value to be stored here
    newNode->next = NULL; // Be helpful to the person calling this code
    return newNode;
    // Don't do what is commented below. Statically declared variables
    // are deleted once the function returns
    // Node newNode;
    // return &newNode;
}

// Add a node at the end of the list and return the head of the list
// Head of the list can change if the list was empty to begin with
Node* insertAtEnd(Node *head, float x){
    if(head == NULL)
        head = createNode(x); // New head is the first node
    else
        head->next  = insertAtEnd(head->next, x);
    return head;
}

// Insert the given data (float x) at specified index of the linked list
// Return address of the head node of the linked list since the head
// node may have changed after if insertion took place at idx 0
Node* insertAtPosition(Node *head, float x, int idx){
    if(idx == 0){ // Insert here
        Node *newHead = createNode(x);
        newHead->next = head;
        head = newHead; // This is the new head now
    }else if(head == NULL){
    // If idx != 0 and head == NULL, then something is wrong. This means
    // the list has only k nodes and I am asking the k+2-th or k+3-th
    // node to be inserted - this is wrong. The k+1-th node should
    // be inserted before k+2-th node is inserted
        printf("ERROR - cant insert at this position in the list\n");
    }
    else{ // All is well - recursively call this routine
        head->next = insertAtPosition(head->next, x, idx-1);
    }

    // Head may not have changed but as promised, return the head
    return head;
}

// Delete the node at specified index of the linked list and return the
// address of the new head node of the linked list since the head node may
// have changed if the head is the only node in the linked list
Node* deleteAtPosition(Node *head, int idx){
    if(head == NULL)
        printf("ERROR - nothing to delete at this position\n");
    else{
        if(idx == 0){ // Delete this node
            Node *newHead = head->next;
            free(head); // Free the head
            head = newHead; // Next becomes new head
        }else{
            head->next = deleteAtPosition(head->next, idx - 1);
        }
```

```
    }
    // Head may or may not have changed but as promised, return head
    return head;
}

// Delete the head and return the new head
Node* deleteHead(Node *head){
    return deleteAtPosition(head, 0);
}

// Delete the entire list
Node* dumpList(Node *head){
    while(head != NULL)
        head = deleteHead(head);
    return head;
}

// Get a pointer to the node in the linked list at index idx
// If idx is illegal, return a NULL pointer
Node* getNodeAtPosition(Node *curr, int idx){
    Node *answer = curr; // If idx == 0, answer is current node itself
    if(curr == NULL)
        printf("ERROR - no such node exists in the list\n");
    else{
        if(idx > 0)
            answer = getNodeAtPosition(curr->next, idx-1);
    }
    return answer;
}

// Move numPos positions right from current position in the linked list
Node* movePositionsRight(Node *curr, int numPos){
    return getNodeAtPosition(curr, numPos);
}

// Traverse and print the whole list
void traverse(Node *head){
    if(head == NULL){
        printf("X\n");
    }else{
        printf("%0.2f => ", head->x);
        traverse(head->next);
    }
}

int main(){
    Node *head = NULL;
    traverse(head);
    head = insertAtEnd(head, 1.0);
    traverse(head);
    insertAtEnd(head, 2.0);
    traverse(head);
    insertAtEnd(head, 3.0);
    traverse(head);
    head = insertAtPosition(head, -1.0, 0);
    traverse(head);
    head = insertAtPosition(head, 10.0, 9);
    traverse(head);
    head = insertAtPosition(head, 1.5, 2);
    traverse(head);
    head = insertAtEnd(head, 4.0);
    traverse(head);
    Node *ptr = getNodeAtPosition(head, 3);
    printf("Node at index 3 is %0.2f\n", ptr->x);
    // Move 2 locations to the right and print the node there
    ptr = movePositionsRight(ptr,1);
    if(ptr != NULL)
        printf("Move 1 location to the right and find %0.2f\n", ptr->x);
    ptr = movePositionsRight(ptr,1);
```

```
    if(ptr != NULL)
        printf("Move 1 more location to the right and find %0.2f\n", ptr->x);
    ptr = movePositionsRight(ptr,1);
    printf("Move 1 more location to the right and find nothing :)\n");
    // Cannot move left in a linked list
    // Need a doubly linked list for that
    head = deleteAtPosition(head, 4);
    traverse(head);
    head = deleteHead(head);
    traverse(head);
    head = deleteAtPosition(head, 4);
    traverse(head);
    head = deleteAtPosition(head, 2);
    traverse(head);
    head = deleteAtPosition(head, 2);
    traverse(head);
    head = dumpList(head);
    traverse(head);
    return 0;
}
```