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

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

typedef struct Node{
    float x;
    struct Node *prev; // The previous node in the list
    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->prev = NULL; // Be helpful to the person calling this code
    newNode->next = NULL; // It is nice to be considerate
    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) // This is the first node
        head = createNode(x);
    // Need to handle the case below separately in case of doubly linked
    // lists since we need to establish back links as well
    else if(head->next == NULL){ // head is the last node of the list
        Node *newNode = createNode(x);
        head->next = newNode; // Forward link
        newNode->prev = head; // Backward link
    }else // There are more nodes to the right in the list
        head->next  = insertAtEnd(head->next, x);
    // As promised, return the head even if it is still the same
    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 this to become new head
        // Need to handle this case specially here because of backlinks
        Node *newHead = createNode(x);
        newHead->next = head; // Fix one forward link
        if(head != NULL){ // If there is a head
            // Fix the other forward link
            if(head->prev != NULL)
                (head->prev)->next = newHead;
            // Fix the back links
            // Warning: do not interchange the next two lines
            // If you do so you will lose the pointer to the old prev
            newHead->prev = head->prev;
            head->prev = newHead;
        }
        head = newHead; // This is the new head now
    }else{ // idx > 0
                if(head == NULL){
        // If idx > 0 and head == NULL, then something is wrong.
        // This means that I am asking an element to be inserted into
        // e.g. k+2-th position when the list has only k elements - 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{ // Okay, there is a head
                        if(idx == 1){ // Insert just after head
                                Node *newNode = createNode(x);
                                newNode->prev = head; // Establish back link
                                if(head->next != NULL) // If there is a next node
```

```
                                        (head->next)->prev = newNode; // The other back link
                                // Fix the forward links now
                                // Warning: do not interchange the next two lines
                                // If you do so you will lose the pointer to the old next
                                newNode->next = head->next; // Insert new node in the middle
                                head->next = newNode;
                                // head remains unchanged
                        }else{ // idx > 1 - 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 address
// of the first node of the linked list since the first node may have
// changed after if the last node of the list gets deleted
Node* deleteAtPosition(Node *head, int idx){
        if(head == NULL)
                printf("ERROR - nothing to delete at this position\n");
        else{
                if(idx == 0){ // Delete the old head
                        Node *newHead = head->next;
                        // Fix the broken back and forward links
                        if(head->prev != NULL)
                                (head->prev)->next = newHead;
                        if(newHead != NULL)
                                newHead->prev = head->prev;
                        head = newHead; // We have a new head
                }else{ // idx > 0
                        head->next = deleteAtPosition(head->next, idx-1);
                }
        }
    // Head may not have changed but as promised, return the 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;
}

// Returns the node idx locations from the current node.
// If idx < 0, returns nodes to the left of the current node
// 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);
        else if(idx < 0)
           answer = getNodeAtPosition(curr->prev, 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);
}
// Move numPos positions left from current position in the linked list
Node* movePositionsLeft(Node *curr, int numPos){
    return getNodeAtPosition(curr, -numPos);
}

void traverse(Node *head){
    if(head == NULL){
        printf("X\n");
    }else{
        printf("%0.2f <=> ", head->x); // <=> symbol since doubly linked
        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 = movePositionsLeft(ptr,3);
    if(ptr != NULL)
        printf("Move 3 locations to the left and find %0.2f\n", ptr->x);
    ptr = movePositionsLeft(ptr,2);
    if(ptr != NULL)
        printf("Move 2 more locations to the left and find %0.2f\n", ptr->x);
    ptr = movePositionsLeft(ptr,2);
    printf("Move 2 more location to the left and find nothing :)\n");
    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;
}
```