```c
/**************************************
Program:       LinkedList.h
date:          30/10/2021
by :           Mike Mico

purpose:       make linked list class
*****************************************/

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

/****************************************

define a node
has integer value x as the data contained
has a pointer pointing to another node

*******************************************/
struct Node
{
    int x;
    struct Node* next;
};

typedef struct Node node;

/****************************************

define a linked list
consists of one node pointer which is the head of the LL

*******************************************/
typedef struct LinkedList
{
    node* head;
}LL;

/****************************************

instantiate linked list
method takes in 1 parameter: a node pointer (head).

*******************************************/
LL* makeLL(node* head)
{
    //allocate space in heap memory for the linked list.
    LL* myList = (LL*) malloc(sizeof(LL));
    //make the head of the linked list the node we passed as a parameter.
    myList->head = head;

    return myList;
}
```

```
/**************************************
Method:        search
date:          30/10/2021
by :           Mike Mico

purpose:       search for element
*****************************************/



/****************************************

takes 2 parameters: Linked list m and an element E
loop through LL searching for element
return 1 if found and 0 if not found

*******************************************/
bool search(LL* m,int E)
{
    //we create a node pointer current which we will use to loop through the LL
    node* current;
    //set current to the first element in the LL
    current = m->head;
    //boolean will be false unless our element E has been found
    bool found=false;
    //current will be incremented so our end case for the while loop is current ==
NULL
    while (current !=NULL)
    {
        //boolean found will be true only if the element is found
        if(current->x == E)
            found=true;
        //make current point to the next node in the LL
        current = current->next;
    }
    return found;
}
```

```
/**************************************
Method:         count
date:           30/10/2021
by :            Mike Mico

purpose:        count number of instances
*****************************************/


/****************************************

takes 2 parameters: linked list m and an element E
loop through LL counting number if times E appears
return 0 if element not in list otherwise return number of appearances

*******************************************/
int count(LL* m,int E)
{
    //set current to the first element in the list
    node* current = m->head;
    //have a counter starting at 0 and increment every time the element in question
is found
    count = 0;
    while (current !=NULL)
    {
        if(current->x == E)
            count++;
        //make current point to the next node in the LL
        current = current->next;
    }
    return count;
}
```

```
/**************************************
Method:        removeE
date:          30/10/2021
by :           Mike Mico

purpose:       delete element
*****************************************/

/****************************************

takes 2 parameters: linked list m and an element E
loop through LL searching for element E.
remove the first instance of E found in LL.

*******************************************/
void removeE(LL* m,int E)
{
    //create node pointer current.
    node* current;
    //set current to the first element in the LL
    current = m->head;

    /*******************************************
    if element is found at the start of the LL
    ******************************************/

    if (current->x == E)
    {
         //make our head node the next in the list (deleting the head)
         m->head=current->next;
         //return because we only want to remove 1 element
         return;
    }

    /*******************************************
    loop through list while not empty and the
    next pointer is not null.
    ******************************************/
    while (current !=NULL && current->next !=NULL)
    {
        //check if the value of the next node is equal to element E
        if(current->next->x == E)
        {
            //make current point to the node pointed to by element E(deleting E)
            current->next = current->next->next;
            //return because we only want to remove 1 element
            return;
        }
        //increment current
        current = current->next;
    }
}
```

```
/**************************************
Method:        removeAll
date:          30/10/2021
by :           Mike Mico

purpose:       delete linked list
*****************************************/
```

```
/****************************************

take in 1 parameter: the linked list pointer m
void method to clear list by making the head NULL

*******************************************/
void removeAll(LL* m)
{
    m->head=NULL;
}
```

```
/**************************************
Method:        frontInsert
date:          30/10/2021
by :           Mike Mico

purpose:       insert at beginning of list
*****************************************/




/****************************************

takes in 2 parameters: linked list pointer m and element E
void method that inserts a node at the beginning of list

*******************************************/
void frontInsert(LL* m,int E)
{
    //make our head this new node.
    m->head = newNode;
    //make the new node pointer point to the old head node.
    newNode->next = m->head;
    //instantiate our new node by making x value equal to E
    m->head->x = E;

    return ;
}
```

```
/**************************************
Method:        printList
date:          30/10/2021
by :           Mike Mico

purpose:       print linked list
*****************************************/



/****************************************

takes in  parameter: linked list pointer m
void method to print the LL

*******************************************/
void printList(LL* m)
{
    //craete node pointer pointing to head of LL
    node* current = m->head;

    //loop through list while current is not null
    while (current !=NULL)
    {
        //print out the int value of each node
        printf("%d ",current->x);
        //increment current
        current = current->next;
    }
    //print a new line for readability.
    printf("\n");
}
```

```
/**************************************
Method:        swap
date:          30/10/2021
by :           Mike Mico

purpose:       swap nodes
*****************************************/


/****************************************

takes in 2 node pointers a and b as parameters
method to swap 2 nodes

*******************************************/
void swap(node* a, node* b)
{
    //create a temporary int to store x value of node a
    int temp =a->x;
    //change x value of a to x value of b
    a->x = b->x;
    //change x value of b to x temp which was a's value
    b->x = temp;
}
```

```
/**************************************
Method:        sort
date:          30/10/2021
by :           Mike Mico

purpose:       sort Linked List
*****************************************/

/****************************************

take in 1 parameter: the linked list pointer m
void method to sort the list

*******************************************/
void sort(LL* m)
{
    //create node pointer current
    node* current;
    //make current point to first element in list
    current = m->head;

    //check if list is empty
    if(m->head == NULL)
        return;

    /****************************************************

    have two loops;
    - the outer loop will be increment up to size
    of list minus 1
    -the inner loop will go from the second element to the end of list

    *******************************************************/


    while (current->next->next != NULL)
    {
        //compare current node with all other following nodes
        while(current->next != NULL)
        {
            //check if any nodes after current are smaller than current
            //by the end of this loop the smallest element will be swapped with
current
            if (current->x > current->next->x)
            {
                //swap node with current
                swap(current,current->next);
            }
            //increment current
            current = current->next;
        }
    }
}
```