LL.h

```
//Eduardo Martinez
//CS211 Assisgnment 5
//LList - header file

#ifndef LL_H
#define LL_H
#include <string>
using namespace std;

typedef int el_t; // el_t is an alias for int


// This declares a new type of structure called node.
// Each node of a linked list will be of this type.
struct node
{
  el_t elem; // element at this node is an integer
  node* next; // a link (pointer) to the next node
};

class LList
{
 private:
  // Data members are:
  node* front; // where the front element of the queue is.
  node* rear; // where the rear element of the queue is.
  // PURPOSE: (private) to handle unexpected errors
encountered by other methods
  void queueError(string msg);

 public:
  LList(); // constructor
  ~LList(); // destructor
  // HOW TO CALL: pass an element to be added
  // PURPOSE: to add element to rear of the queue
  void addRear(el_t el);
  // PURPOSE: to Remove the front element of queue
  el_t deleteFront();
  //HOW TO CALL:retursn true if the list is empty
  // PURPOSE: to check if the queue is empty
  bool isEmpty();
  //HOW TO CALL: outputs all the elements in the linked
list
  // PURPOSE: to display all elements in the queue
  void displayAll();
  //HOW TO CALL: outputs all elements in linked list
  //in reverse order
  // PURPOSE: to output elements in reverse order
  void printAllReverse();
  // PURPOSE: to output elements in reverse order
  void printAllReverse(node* p);
  //HOW TO CALL:pass an element to be added to the front
//PURPOSE: to add element to the front
  void addFront(el_t el);
  //HOW TO CALL: returns deleted rear
  //Purpose: to remove rear node
  el_t deleteRear();

};

#endif
```

LL.C

```
//Eduardo Martinez
//CS211 Assignment 5
//LL Class - Implementation File
#include "LL.h"
#include <iostream>
using namespace std;

// PURPOSE: constructor which initializes top
LList::LList()
{
  count = 0;
  front = NULL;
  rear = NULL;

}
// PURPOSE: destructor- does nothing
LList::~LList()
{
  while(!isEmpty())
    deleteFront();
}
// PURPOSE: to add element to rear of the queue
// PARAMS: new element el of type el_t
// ALGORITHM: adds element to rear of queue,
void LList::addRear(el_t el)
{
  if(count != 0)
    {
      rear->next = new node; // make the rear node point to
a new node
      rear = rear->next; // rear points to the new one
    }
  else
    front = rear = new node;

  rear->elem = el; // the last node points to nothing
  rear->next = NULL;
  count++;
}
// PURPOSE: to Remove the front element of queue
// ALGORITHM: deletes and returns front, next node becomes
fron
el_t LList::deleteFront()
{
 node* second;
 el_t ch= front->elem;
  if(isEmpty())
    queueError("Error: list is empty");

 second = front->next; // front's next pointer is saved
 delete front; // front node is gone
 front = second; // front pointer points to the new front
node.
 count--;
 return ch;  // what's in the front node?

}
// PURPOSE: to check if the queue is empty
// ALGORITHM: if count = 0 then returns true
bool LList::isEmpty()
{
  if(count == 0)
    return true;
  else
    return false;
}

// PURPOSE: to display all elements in the queue
// ALGORITHM: displays element and points to next node
void LList::displayAll()
{
  //  if(isEmpty())
  //queueError("queue is empty");

  node* p = front;
  while(p != NULL)
    {
      cout << (el_t)p->elem ;
      p=p->next;
    }
}
// PURPOSE: (private) to handle unexpected errors
encountered by other methods
// PARAMS: a string message to be displayed
// ALGORITHM: simply cout the message and exit from the
program
void LList::queueError(string msg)
{
  cout << msg << endl;
  //  exit(1);
}
// PURPOSE: to output elements in reverse order
// ALGORITHM: recursive function that outputs elements in
reverse
void LList::printAllReverse()
{
  printAllReverse(front);
}
// PURPOSE: to output elements in reverse order
// ALGORITHM: returns if p is pointing at NUll, outputs
element
void LList::printAllReverse (node* p)
{
 if(p == NULL)
    return;
  else
    {
      printAllReverse (p->next);
      cout << (el_t)p->elem;
    }
}

void LList::addFront(el_t e)
{
  node* newFront;
  if(isEmpty())
    addRear(e);
  else
    {
      newFront->next= front;
      front = newFront;
      Front->elem = e;
    }
}
el_t LList::deleteRear()
{
  if(isEmpty())
    queueError("queue is empty");
  else
    {
      node* pre;
      node* del;
      pre = front;
      del = front->next;
      for(int i =0; i<=count ;i++)
      {
        if(count == 1)
          deleteFront();

        if(del->next == NULL)
          {
            el_t el = rear->elem;
            delete rear;
            rear = pre;
            rear->next = NULL;
            return el;
          }

            pre = pre->next;
            del= del->next;

      }
    }
```

LLClient.C

```
#include <iostream>
#include "LL.h"
using namespace std;

int main()
{
  LList l;

 l.addFront(1);
 l.addFront(2);
 l.addFront(3);
 l.addRear(4);
 l.addRear(5);

 l.displayAll();
 cout << endl;

 cout << l.deleteFront() << " has been deleted" << endl;
 cout << l.deleteRear() <<  " has been deleted" << endl;

 l.displayAll();
 cout << endl;

 cout << l.deleteFront() << " has been deleted" << endl;
 cout << l.deleteRear() <<  " has been deleted" << endl;

 l.displayAll();
 cout << endl;

  cout << l.deleteFront() << " has been deleted" << endl;

  l.displayAll();
 cout << endl;

 l.addRear(10);
 l.addFront(11);

 l.displayAll();
 cout << endl;

 cout << l.deleteRear() <<  " has been deleted" << endl;
 cout << l.deleteRear() <<  " has been deleted" << endl;

 l.displayAll();
 cout << endl;

 return 0;
}
```

LLCLient.C Test Run:

```
[marti540@empress cs211]$ ./a.out
3 2 1 4 5
3 has been deleted
5 has been deleted
2 1 4
2 has been deleted
4 has been deleted
1
1 has been deleted
queue is empty

11 10
10 has been deleted
11 has been deleted
queue is empty

[marti540@empress cs211]$
```

palindrome.C

```
//Eduardo Martinez
//CS 211 - Assignment 6
//Palindrome.C - check if a string is a palindrome.The
program getsa string from
//the user, makes a linked list from the string(only adding
char), then deletes and
//check if front and rear are equaluntil linked list is
empty. if front and rear are
// not equal, the funtcion returns false
#include <iostream>
#include <string>
#include <cstring>
#include <stdlib.h>
#include "LL.h"
using namespace std;

void createLL(LList& l,string s);
bool palindrome(LList& l);
int main()
{
  LList l;//linked list
  string s;
  bool isPal;//bool for it is a palindrome or not
  cout << "********PALINDROME CHECKER********" << endl;
  cout << "Enter Palindrome:" <<endl;
  getline(cin,s);//gets input
  createLL(l,s);//creates the linked list from the user

  isPal=palindrome(l);
  if(isPal)//if it is a palindrome
    {
      cout << endl;
      cout << " Yea, its a palindrome" << endl;
    }
  else//if not
    {
      cout << endl;
      cout << "Noo, its not a palindrome!" << endl;
    }


  return 0;
}
void createLL(LList& l,string s)
{
  char c[s.length()+1];//create cstring
  strcpy(c,s.c_str());//copy stringt to the cstring

  for(int i=0; i<= s.length();i++)
    {
      int x=(int)c[i];
      if(x>96 && x<123)//if x is a lowercase integer
      x -= 32;//makes it uppercase
      if(x>54 && x<133)
      l.addRear(x);//adds to link list only if its an
uppercase char
    }
}

bool palindrome(LList& l)
{
  bool pal = true;
  while(!l.isEmpty())//while it not empty
    {
      int a= l.deleteFront();
      if(l.isEmpty())//if empty the link list must only
have 1 element
      {
        //do nothing
      }
      else
      {
        if(a != l.deleteRear())//checks if front and rear
are the same
          pal = false;//if not
      }

    }
  return pal;
}
```

Palindrome.C test Runs:

```
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
Racecar

 Yea, its a palindrome
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
Race car

 Yea, its a palindrome
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
Pop

 Yea, its a palindrome
```

```
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
Ada

 Yea, its a palindrome
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
Anna

 Yea, its a palindrome
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
A Santa at Nasa.

 Yea, its a palindrome
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
A Toyota! Race fast, safe car! A Toyota!

 Yea, its a palindrome
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
abcdba

Noo, its not a palindrome!
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
Santa at Nasa

Noo, its not a palindrome!
[marti540@empress cs211]$ ./a.out
********PALINDROME CHECKER********
Enter Palindrome:
as in nasa

Noo, its not a palindrome!
[marti540@empress cs211]$
```