```
//Eduardo Martinez
//CS211 Assignment 5
//Pointer Lab #2

#include <iostream>
using namespace std;

int main()
{
  int* p1;
  int* p2;

  p1= new int;
  *p1=1;
  cout << *p1 << endl;

  p2= new int;
  *p2=2;
  cout << *p2 << endl;

  p2 = p1;
  cout << *p2 << endl;

  delete p2;
  p1 = NULL;

  p2= new int;
  *p2 = 3;
  cout << *p2 << endl;

  return 0;
}
```

```
//Eduardo Martinez
//CS211 Assisgnment 5
//LQueue class - header file

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

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


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

class LQueue
{
 private:
  // Data members are:
  int count; // how many elements do we have right now?
  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:
  LQueue(); // constructor
  ~LQueue(); // 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();
  // PURPOSE: to check if the queue is empty
  bool isEmpty();
  // PURPOSE: to display all elements in the queue
  void displayAll();
  // PURPOSE: to output elements in reverse order
  void printAllReverse();
  // PURPOSE: to output elements in reverse order
  void printAllReverse(node* p);
};
#endif
```

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

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

}
// PURPOSE: destructor- does nothing
LQueue::~LQueue()
{
  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 LQueue::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 LQueue::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 LQueue::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 LQueue::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 LQueue::queueError(string msg)
{
  cout << msg << endl;
  //  exit(1);
}
// PURPOSE: to output elements in reverse order
// ALGORITHM: recursive function that outputs elements in reverse
void LQueue::printAllReverse()
{
  printAllReverse(front);
}
// PURPOSE: to output elements in reverse order
// ALGORITHM: returns if p is pointing at NUll, outputs element
void LQueue::printAllReverse (node* p)
{
 if(p == NULL)
    return;
  else
    {
      printAllReverse (p->next);
      cout << (el_t)p->elem;
    }
}
```

<u>OutPut From LQueue CLient</u>

```
[marti540@empress cs211]$ g++ lqueue.C lqueueClient.C
[marti540@empress cs211]$ ./a.out
 is empty

1
1234
1
2
3
4
 is not empty
4
 is empty

[marti540@empress cs211]$
```

```
//Eduardo Martinez
//CS211 Assignment5
//LQueue Client File
//the purpose of the lab is to add 2 large int numbera
#include <iostream>
#include <string>
#include "lqueue.h"
#include <stdlib.h>
using namespace std;

//creates in reverse an lqueue with one number in each node
void createReverseLL(LQueue& l, string s);
//adds both l1 and l2 and sum goes into total
void addLLs(LQueue& total,LQueue& l1,LQueue& l2);
int main()
{
  LQueue l1,l2,total;//create lqueue objects
  string s1,s2;//create string


  cout << "enter a positive integer" << endl;
  cin >> s1;//user inputs int
  createReverseLL(l1,s1);
  cout << "enter a positive integer" << endl;
  cin >> s2;//user inputs int
  createReverseLL(l2,s2);//creates and reverses

  l1.displayAll();//display all the numbers in l1
  cout << " + ";
  l2.displayAll();//display all the number in l2
  cout << " = ";
  addLLs(total,l1,l2);//adds l1 and l2 together, sum goes into total

  total.displayAll();// displays the total in reverse
  cout  << endl;


  createReverseLL(l1,s1);
  createReverseLL(l2,s2);
  l1.printAllReverse();//prints l1 in reverse order
  cout << " + ";
  l2.printAllReverse();//prints l2 in reverse order
  cout << " = ";
  total.printAllReverse();// prints total in reverse order
  cout << endl;
  return 0;

}
void createReverseLL(LQueue& l, string s)
{
  char cstr[s.length()+1];//creates a c-string
  strcpy (cstr, s.c_str());//copies the string int cstr

  for(int x= s.length()-1; -1 < x; x--)
    {
      l.addRear((cstr[x]-48));//adds the int value to rear of queue
    }

}
void addLLs(LQueue& total,LQueue& l1,LQueue& l2)
{
  int r=0;//
  int sum,n1,n2;

  while(!l1.isEmpty() || !l2.isEmpty())
    {
      if(l1.isEmpty())//l1 is empty
        {
          n1=0;//l1 is empty
          n2=l2.deleteFront();//l2 is not empty
        }
      else if(l2.isEmpty())//l2 is empty
        {
          n1 =l1.deleteFront();//l1 is not empty
         n2=0;//l2 is empty
        }
      else if(!l1.isEmpty() && !l2.isEmpty())
        {
          n1 =l1.deleteFront();//l1 is not empty
          n2=l2.deleteFront();//l2 is not empty
        }

      sum=n1+n2+r;//sum of l1, l2, and carry over
      r=sum/10;//if sum >= 11 then 1 is carried over to next sum
      total.addRear(sum%10);// number is added to rear of total
    }
  if(r>0)//if the last sum is >=10
        total.addRear(r);

}
```

Output from add2LargeInt.C

```
[marti540@empress cs211]$ g++ lqueue.C add2LargeInt.C
[marti540@empress cs211]$ ./a.out
enter a positive integer
1
enter a positive integer
```

```
1
1 + 1 = 2
1 + 1 = 2
[marti540@empress cs211]$ ./a.out
enter a positive integer
111
enter a positive integer
111
111 + 111 = 222
111 + 111 = 222
[marti540@empress cs211]$ ./a.out
enter a positive integer
191
enter a positive integer
131
191 + 131 = 223
191 + 131 = 322
[marti540@empress cs211]$ ./a.out
enter a positive integer
191
enter a positive integer
931
191 + 139 = 2211
191 + 931 = 1122
[marti540@empress cs211]$ ./a.out
enter a positive integer
917
enter a positive integer
47
719 + 74 = 469
917 + 47 = 964
[marti540@empress cs211]$ ./a.out
enter a positive integer
999
enter a positive integer
8
999 + 8 = 7001
999 + 8 = 1007
[marti540@empress cs211]$ ./a.out
enter a positive integer
999
enter a positive integer
8888
999 + 8888 = 7889
999 + 8888 = 9887
[marti540@empress cs211]$ ./a.out
enter a positive integer
999
enter a positive integer
9888
```

```
999 + 8889 = 78801
999 + 9888 = 10887
[marti540@empress cs211]$ ./a.out
enter a positive integer
999
enter a positive integer
99999
999 + 99999 = 899001
999 + 99999 = 100998
[marti540@empress cs211]$ ./a.out
enter a positive integer
22222222222222222
enter a positive integer
33333333333333339
22222222222222222 + 93333333333333333 = 16555555555555555
22222222222222222 + 33333333333333339 = 55555555555555561
[marti540@empress cs211]$
```