CS 162 – Final Exam
DUE: Wednesday, 6/10/2015, at 11:59pm
(No late work accepted!!!)

Your final exam in this class is a take home project, and it is to be completed on your own (not as a group!). Below is C code for the merge sort algorithm. It is your job to produce a main.c that tests the functionality of the mergeSort library and to explain exactly what the code is doing under different test cases.

merge.h - struct and function declarations

```
struct node {
   int info;
   struct node *next;
};
void divideList(struct node *,struct node **);
struct node * mergeList(struct node *,struct node *);
void recMergeSort(struct node **);
void mergeSort(struct node **, struct node **);
```

merge.c - function definitions

```
void divideList(struct node* first1,struct node** first2)
{
   struct node* middle;
   struct node* current;
   if (first1 == NULL)  //list is empty
      *first2 = NULL;
   else if (first1->next == NULL)  //list has only one node
      *first2 = NULL;
   else
   {
      middle = first1;
      current = first1->next;
      if (current != NULL)     //list has more than two nodes
         current = current->next;
      while (current != NULL)
      {
         middle = middle->next;
         current = current->next;
         if (current != NULL)
            current = current->next;
      } //end while
      *first2 = middle->next;
      middle->next = NULL;
   } //end else
} //end divideList
```

```
struct node* mergeList(struct node* first1,struct node* first2)
{
   struct node *lastSmall; //pointer to the last node
   struct node *newHead;  //pointer to the merged list

   if (first1 == NULL)  //the first sublist is empty
      return first2;
   else if (first2 == NULL)  //the second sublist is empty
      return first1;
   else
   {
      if (first1->info < first2->info) //compare the first nodes
      {
         newHead = first1;
         first1 = first1->next;
         lastSmall = newHead;
      }
      else
      {
         newHead = first2;
         first2 = first2->next;
         lastSmall = newHead;
      }
      while (first1 != NULL && first2 != NULL)
      {
         if (first1->info < first2->info)
         {
            lastSmall->next = first1;
            lastSmall = lastSmall->next;
            first1 = first1->next;
         }
         else
         {
            lastSmall->next = first2;
            lastSmall = lastSmall->next;
            first2 = first2->next;
         }
      } //end while
      if (first1 == NULL) //first sublist is exhausted first
         lastSmall->next = first2;
      else      //second sublist is exhausted first
         lastSmall->next = first1;
      return newHead;
   }
}//end mergeList
```

```
void recMergeSort(struct node** head)
{
   struct node* otherHead;

   if (*head != NULL)  //if the list is not empty
      if ((*head)->next != NULL)  //if list has more than one
      {
         divideList(*head, &otherHead);
         recMergeSort(head);
         recMergeSort(&otherHead);
         *head = mergeList(*head, otherHead);
      }
} //end recMergeSort

void mergeSort(struct node **first, struct node **last)
{
   recMergeSort(first);

   if (*first == NULL)
      *last = NULL;
   else
   {
      *last = *first;
      while ((*last)->next != NULL)
      *last = (*last)->next;
   }
} //end mergeSort
```

Your paper must include:

- **A table** with **at least 10 more test cases** and results to make sure the `mergeSort` function works. Think about the number of nodes and the initial values you want to use for testing. **You need to have a table with these headings for your testing:**

Input Values	Expected Output	Did Actual Meet Expected?
Empty List, head and tail are NULL	nothing	Yes
…		

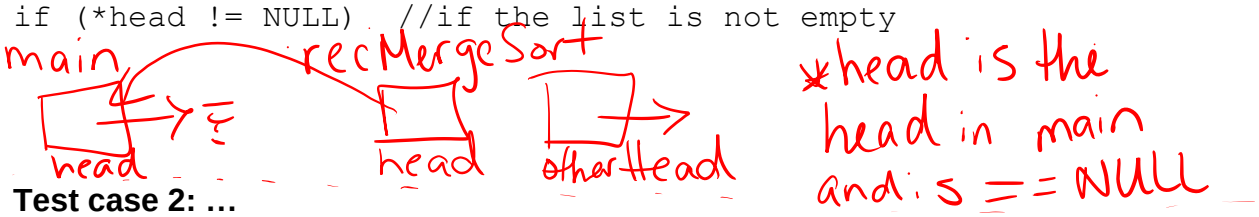
- **Description** of how `void recMergeSort(struct node** head)` works. You need to **describe each line of the function for four of your test cases** you have. **This should include pictures and text**. If you don't have a tablet, you can take pictures of your drawings or scan them in as pictures to **embed in the document**.

  Because `recMergeSort` makes a call to `divideList`, 2 recursive calls to itself, `recMergeSort`, and a call to `mergeList`, you must describe what happens in these 4 function calls. **For the recursive calls**, you must label the

depth and which recursive call (first or second) that you are describing or drawing the picture for.

Test case 1:
When the list is empty, *head (which is the head pointer to the list) in `recMergeSort` is NULL and the first if checks to see if *head is NULL, i.e.

```
if (*head != NULL)  //if the list is not empty
```



Test case 2: …
Test case 3: …
Test case 4: …

- **Breakdown of points/grading:**
  10 pts. Testing
  The table for 10 additional test cases (you will be graded on what you chose to make sure it worked, i.e. how many nodes in the list and what were the initial values)
  80 pts. Displaying understanding of the merge sort algorithm
  20 pts. Description and pictures of your first test case
  20 pts. Description and pictures of your second test case
  20 pts. Description and pictures of your third test case
  20 pts. Description and pictures of your fourth test case
  10 pts. Using a library (.h and .c files)
  `main.c` file that tests the `mergeSort` algorithm.

The purpose of this final paper is to convince me (and yourself☺) how the `mergeSort` algorithm works. The hardest part of this algorithm is understanding the recursive nature of the solution and how it works. Write this document as if you were writing a tutorial for a student trying to learn and understand how merge sort works!!!

You must turn in your **main.c file** that tests the functionality of the mergeSort algorithm and a **paper as a .pdf or .doc (pictures embedded in the document)** to the TEACH website by the due date above. **No late finals accepted!!!!**