NAME (print): _____

Honor Acknowledgment (signature): _____

DO NOT SPEND MORE THAN 10 MINUTES ON ANY OF THE OTHER QUESTIONS! If you don't see the solution to a problem right away, move on to another problem and come back to it later.

Before starting, make sure your test contains 7 pages.

If you think there is a syntax error, then ask. You may assume any include statements are provided.

	value	grade
Problem 1	6 pts.	
Problem 2	16 pts.	
Problem 3	12 pts.	
Problem 4	20 pts.	
Problem 5	12 pts.	
TOTAL:	72 pts.	

Whenever you see references to *Node* in this test, the following definition is assumed.

```
template <class T>
struct Node
{
    T info;
    Node<T> * next;

    Node(const T & newInfo, Node<T> * newNext = NULL)
        : info(newInfo),
          next(newNext)
    {}
};
```

PROBLEM 1 : (*Around and around ...* (6 points))

What is the running time T(n) as expressed by using O(?) (or big O) for the function *mystery* shown below?

```
double mystery(int n)
{
    int j, k;
    double ans = 0.0;
    k = n;
    while (k > 0)
    {
        k /= 2;
        for (j = 1; j <= n; j++)
        {
            ans += k * (j * j - 1);
        }
    )
    return ans;
}
```

PROBLEM 2 : **(***Locate and Apprehend* (16 points)**)**

Part A (*8 points*)

Write the function *getNode* whose header is shown below. The function *getNode* locates a node by matching the node's info field with the given key. It then removes that node from the list and returns a pointer to it. If there are multiple occurrences that match the key in the nodes of list, the first node matching should be processed.

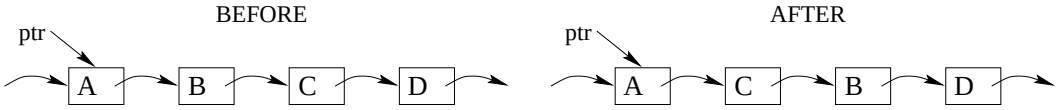
If the node is not found in list, *getNode* should return NULL.

```
Node<string> * getNode(Node<string> * & list, string key)
// pre: list is NULL terminated linked with nodes of type Node shown elsewhere.
//      key is a string which is to be matched (if possible) by the info
//      field of a node.
// post: If key is matched in a node, then that node is removed from list and
//      a pointer to that node is returned.  Else NULL is returned.
{
```

What is the worst case running time T(N) for *getNode* as expressed by O(?) (or big O) in relation to the number of nodes, N, in the list?

Part B (*8 points*)

Write the function *xchangeNode* whose header is shown below. *xchangeNode*, given a pointer to a node in a linked list, interchanges the two following nodes. If there are not at least two nodes following the node pointed to, then the function should not change the list.




```
void xchangeNode(Node<string> * ptr)
// pre: ptr points to a node in a null terminated linked list.
// post: the two nodes following the one pointed to by ptr have been reversed
//       in order.  If there are not at least two nodes after the node pointed
//       to by ptr, the function does nothing.
{
```

PROBLEM 3 : (*Fair is fair* (12 points))

Write the function *mergeByAlternating* whose header is shown below. Given two linked lists, *listA* and *listB,* merge their nodes into a single list. Alternate the source of the nodes, taking the first one from listA, the second one from listB, the third one from list A, etc. The function *mergeByAlternating* should return a pointer to the merged list. If the lists are not of the same length, simply add the nodes in the remaining list to the merged list. Note that no nodes should be created or destroyed in this process.

```
template <class T>
Node<T> * mergeByAlternating(Node<T> * & listA, Node<T> * & listB)
// pre: listA and listB are pointers to null terminated linked list. Either
//      or both could be empty.
// post: a pointer to a merged linked list is returned consisting of
//      alternately nodes from listA and listB plus the tail of the longer
//      list if they are unequal in length.  listA and listB are empty.
{
```

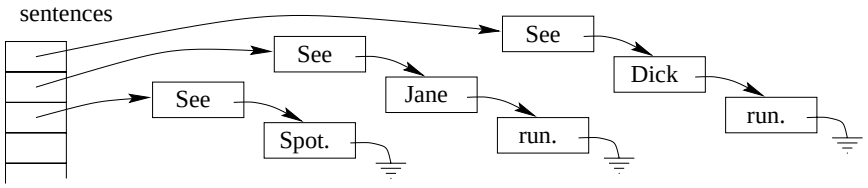
PROBLEM 4 : (*Words Storage* (20 points))

Write the function *buildSentences* whose header is shown below, so that it takes a text file and reads the words in. Each sentence (assume they are all terminated by periods and there are no other periods) should end up in a separate linked list. The pointers to these linked lists are contained in a vector. Assume that the given vector is large enough to hold all of the sentences in the file. When done, the function should have updated the *numSentences* parameter to tell how many sentences were read (i.e. what portion of the vector has been used).

Assume that the function is passed an open file, an vector, and a size parameter. You can also make use of the function *hasPeriod* to determine if a given word contains a period. Its header is shown below (do not write this).

```
bool hasPeriod(const string & word);
// returns true if word contains a period, false otherwise.
```

For example, a file containing the words: *See Dick run. See Jane run. See Spot.* Should result in the following:




```
void buildSentences(istream & input, Vector<Node<string> *> sentences,
                                                        int & numSentences)
// pre: input is open istream, sentences is a vector of pointers to nodes,
//      all initialized to NULL.  numSentences is the number of sentences
//      currently pointed to by the vector.
// post: input is at end of file, The first numSentences elements of
//      sentences each point to a null terminated linked list. The info
//      fields of each node in a linked list contain successive words in
//      the file.  The last node in each linked list contains a word with
//      a period at its end.
{
```

PROBLEM 5 : (*Divide and conquer* (12 points))

Consider the following function, *sum,* which recursively adds together the elements in a vector with N elements where N is a power of 2.

```
int sum(Vector<int> data, int n, int m)
// pre: data is a vector of integers, n <= m
// post: sum returns the sum of elements n thru m
{
    if (n == m)
        return data[n];
    else
        return sum(data, n, n + (m-n)/2) + sum(data, n + (m-n)/2 + 1,m);
}
```

The following recurrence relation describes the algorithm implemented. Solve the recurrence relation.

$T(1) = 1$

$T(N) = 2T(N/2) + C$