Prepare your answers to the following questions in a plain text file. Submit your file to the Curator system by the posted deadline for this assignment. No late submissions will be accepted.

You will submit your answers to the Curator System ([www.cs.vt.edu/curator](http://www.cs.vt.edu/curator)) under the heading Homework1.

Both of these questions refer to the DListT template interface given as part of Minor Project 1. Make sure that your solutions take that into account.

1. [20 pts] Write an implementation for a client (i.e., non-member, non-friend) template function that will take two DListT objects (instantiated on the same type, obviously) and return a new DListT object that is the symmetric difference of the two parameters (i.e., it contains one copy of each value that occurs the first list but not in the second list, and one copy of each value that occurs in the second list but not in the first list). For example, the symmetric difference of these two lists

\{4, 17, 3, 2, 5\} and \{3, 1, 4, 2\}

would be the list: \{1, 5, 17\}. The order of the elements in the symmetric difference list does not matter.

Your solution may assume that neither of the lists it receives contains more than one occurrence of the same value. Your solution should conform to the interface below:

```cpp
template <typename T>
DListT<T> SymmetricDifference(const DListT<T>& L1, const DListT<T>& L2) {
    DListT<T> SymmDiff;  // create empty list to hold symmetric difference
    typename DListT<T>::const_iterator Curr1  = L1.begin();
    typename DListT<T>::const_iterator Stop1  = L1.end();
    typename DListT<T>::const_iterator Curr2  = L2.begin();
    typename DListT<T>::const_iterator Stop2  = L2.end();
    for (; Curr1 != Stop1; Curr1++) {  // add L1 – L2 to symm. difference
        if ( L2.Find(*Curr1) == Stop2 )
            SymmDiff.Insert(SymmDiff.end(), *Curr1);
    }
    for (; Curr2 != Stop2; Curr2++) {  // add L2 – L1 to symm. difference
        if (L1.Find(*Curr2) == Stop1 )
            SymmDiff.Insert(SymmDiff.end(), *Curr2);
    }
    return SymmDiff;
}
```
2. [20 pts] Write an implementation for a `DListT` template member function that returns two new lists obtained by splitting the original list (roughly) into front and back halves. The front half list should contain the “extra” element if the original list contained an odd number of elements.

Your solution must not make any assumptions about the number of elements in the original list, and the original list must not be modified. Your solution should make only one pass through the elements in the list. Your solution should conform to the interface below:

```cpp
template <typename T>
void DListT<T>::Split(DListT<T>& frontHalf, DListT<T>& backHalf) const {
    frontHalf.Clear(); // make sure supplied lists are empty
    backHalf.Clear();
    if (Head == NULL) return; // if list is empty, nothing to do

    DNodeT<T>* Front = Head;
    DNodeT<T>* Back = Tail;

    // walk pointers from head and tail until they converge or pass
    while (Front != Back && Front->Next != Back) {
        frontHalf.Insert( frontHalf.end(), Front->Element );
        backHalf.Insert( backHalf.begin(), Back->Element );
        Front = Front->Next;
        Back = Back->Prev;
    }

    // determine which termination case we have (even or odd # of elems)
    // and take appropriate action
    if (Front == Back)
        frontHalf.Insert( frontHalf.end(), Front->Element );
    else {
        frontHalf.Insert( frontHalf.end(), Front->Element );
        backHalf.Insert( backHalf.begin(), Back->Element );
    }
}
```