CMPT135 J. Ye

Week 7 Exercise Question 2 to 4 Solution

```
#include <iostream>
using namespace std;

// Linked List with Node class

class Node
{
public:
    Node () : data(0), link(NULL) {}
    Node (int theData, Node* theLink) : data(theData), link(theLink) {}
    Node* getLink () const { return link; }
    int getData () const { return data; }
    void setData (int theData) { data = theData; }
    void setLink (Node* pointer) { link = pointer; }
private:
    int data;
    Node* link;
};
typedef Node* NodePtr;


void head_insert (NodePtr& head, int theData)
// Precondition: The pointer variable head points to the head of a linked list.
// Postcondition: A new node containing theData has been added at the head of the linked list.
{
     head = new Node(theData, head);
}


void insert (NodePtr afterMe, int theData)
// Precondition: afterMe points to a node in a linked list.
// Postcondition: A new node containing theData has been added after the node pointed to
// by afterMe.
{
     afterMe->setLink (new Node(theData, afterMe->getLink()));
}


NodePtr search (NodePtr head, int target)
// Precondition: The pointer head points to the head of a linked list. The pointer variable
// in the last node is NULL. If the list is empty, then head is NULL.
// Returns a pointer that points to the first node that contains the target. If no node
// contains the target, the function returns NULL.
{
     NodePtr here = head;

     if (here == NULL) //if empty list
          return NULL;

     else
     {
          while (here->getData() != target && here->getLink() != NULL)
               here = here->getLink();

          if (here->getData() == target)
               return here;
          else
               return NULL;
     }
}
```

```
void remove (NodePtr& head, int num) // remove all the nodes that contain the value num
{
    NodePtr before = head;
    NodePtr discard = head;
    while (discard != NULL)
    {
        if (discard->getData() != num)
        {
            before = discard;
            discard = discard->getLink();
        }
        else if (discard == head)
        {
            head = head->getLink();
            before = head;
            delete discard;
            discard = head;
        }
        else
        {
            before->setLink(discard->getLink());
            delete discard;
            discard = before->getLink();
        }
    }
}


void print_list (NodePtr head)
{
    for ( NodePtr iter = head; iter != NULL; iter = iter->getLink() )
        cout << iter->getData() << " ";
    cout << endl;
}


NodePtr mergeLists ( NodePtr& head1, NodePtr& head2)
{
    NodePtr head, follow, smaller; // head points to the merged list

    // If one list is NULL, return the other head.
    if ( head1 == NULL && head2 != NULL )
    {
        head = head2;
        head2 = NULL;
        return head;
    }
    if ( head1 != NULL && head2 == NULL )
    {
        head = head1;
        head1 = NULL;
        return head;
    }
    if ( head1 == NULL && head2 == NULL )
        return NULL;

    // Now, both head1 and head2 are not NULL, get one node attached to head...
    if ( head1->getData() < head2->getData() )
    {
        head = head1;// smallest node attached to head
        head1 = head1->getLink();
    }
    else
    {
        head = head2;   // as above, for head2
        head2 = head2->getLink();
    }

    follow = head;

    // Connect the rest of them.
    while ( head1 != NULL && head2 != NULL )
    {
        if ( head1->getData() < head2->getData() )
        {
            smaller = head1;
            head1 = head1->getLink();
        }
        else
        {
            smaller = head2;
            head2 = head2->getLink();
        }

        follow->setLink(smaller); // don't forget this!
        follow = smaller;
    }

    // At this point, one of head1 or head2 is NULL, the list is run out.
    // All you need to do is to connect the rest of the other list.
    if( head1 != NULL )
        follow->setLink(head1);

    if ( head2 != NULL )
        follow->setLink(head2);

    head1 = NULL;
    head2 = NULL;

    return head;
}


int main() // The testing code for Week 7 Exercise Q2 and Q3 is the same as Q1's testing code, ignored.
{
    // The following is to test the mergeLists function:
    NodePtr list1 = NULL;
    NodePtr list2 = NULL;

    for (int i = 13; i > 0; i -= 3)
        head_insert (list1, i);

    for (int i = 8; i >= 2; i -= 2)
        head_insert (list2, i);

    print_list (list1);
    print_list (list2);

    NodePtr merged = mergeLists (list1, list2);

    print_list (list1);
    print_list (list2);
    print_list (merged);

    return 0;
}
```