makefile

```
###################################################################
# Program:
#     Week 11, Sorts
#     Brother Ercanbrack, CS235
# Author:
#     Yurii Vasiuk
# Summary:
#     This program will complete a heap merge and merge sort
#     on files filled with intergers.
# Time:
#     15 hours
###################################################################

###################################################################
# The main rule
###################################################################
a.out: week11.o
       g++ -o a.out week11.o
       tar -cf week11.tar *.h *.cpp makefile

###################################################################
# The individual components
#      week11.o     : the driver program
###################################################################
week11.o: merge.h heap.h week11.cpp
       g++ -c week11.cpp
###################################################################
# Makes clean
###################################################################
clean:
       rm a.out *.o *.tar
all : a.out
```

90/100

Commented [ES1]:
1. Merge Sort hangs!
2. Command Line arguments don't work

-10 pts

heap.h

```
/**********************************************************************
* Program:
*    Week 11, Sorting
*    Brother Ercanbrack, CS 235
*
* Author:
*   Yurii Vasiuk
* Summary:
*   The implementation of the heap sort
**********************************************************************/
#include <vector>

using namespace std;

/**********************************************************************
* Percolate down
**********************************************************************/
template<class T>
void percolateDown(vector<T> & data, int first, int last)
{
   T temp;
   int child = first * 2;
   while (child < last)
   {
      if (child < last - 1 && data[child] < data[child + 1])
         child++;
      if (data[first] < data[child])
      {
         temp = data[first];
         data[first] = data[child];
         data[child] = temp;
         first = child;
```

```
         child = first * 2;
      }
      else
         break;
   }
}

/*************************************************************************
* Heapify the tree
*************************************************************************/
template<class T>
void heapify(vector<T> & myHeap)
{
   for (int i = myHeap.size() / 2 - 1; i > 0; i--)
   {
      percolateDown(myHeap, i, myHeap.size());
   }
}

/*************************************************************************
* This function sorts a vector using a heap sort.
* Input:  data -  Vector to be sorted.
* Output: data -  Vector sorted
*************************************************************************/
template<class T>
void heapSort(vector<T> & data)
{
   heapify(data);

   T temp;
   for (int i = (data.size() - 1); i > 1; i--)
   {
      temp = data[1];
      data[1] = data[i];
      data[i] = temp;
      percolateDown(data, 1, i);
   }
}
```

merge.h

```
/*************************************************************************
* Program:
*    Week 11, Sorting
*    Brother Ercanbrack, CS 235
*
* Author:
*   Yurii Vasiuk
* Summary:
*   The implemetation of the merge sort
*************************************************************************/
#include <list>

using namespace std;

/*************************************************************************
* This function will be used in the merge sort
*************************************************************************/
template<class T>
bool split(list<T> & data, list<T> & list1, list<T> & list2)
{
   typename list<T>::iterator itF = data.begin();
   typename list<T>::iterator itS = ++data.begin();
   int numElList1 = 0;

   // till the end of the data list
   while (itS != data.end())
   {
      // filling the first sublist
      list1.push_back(*itF);
      numElList1++;
      while (itS != data.end() && *itF < *itS)
      {
         list1.push_back(*itS);
         itF++;
         itS++;
         numElList1++;
      }
```

```
      if (itS != data.end())
      {
         itF++;
         itS++;
      }
      // already sorted, return true
      if (numElList1 == data.size())
         return true;

      // filling the second sublist
      list2.push_back(*itF);
      while (itS != data.end() && *itF < *itS)
      {
         list2.push_back(*itS);
         itF++;
         itS++;
      }
      if (itS != data.end())
      {
         itF++;
         itS++;
      }
   }

   // the data is not sorted yet, there will be more splitting
   return false;
}

/*************************************************************************
* This function will be used in the merge sort
*************************************************************************/
template<class T>
void merge(list<T> & data, list<T> & list1, list<T> & list2)
{
  // THIS METHOD WORKS IN THE VISUAL STUDIO BUT THE LINUX LAB DOES NOT UNDERSTAND next() AND DOES NOT COMPILE, SO I
 COMMENTED OUT THE WHOLE THING
  /*
   // for list1
   typename list<T>::iterator it11 = list1.begin();   // the beginning of the sublist
   typename list<T>::iterator it12 = list1.begin(); // the next after the end of the sublist
   // for list2
   typename list<T>::iterator it21 = list2.begin();   // the beginning of the sublist
   typename list<T>::iterator it22 = list2.begin(); // the next after the end of the sublist

   // until the end of the one of the lists
   while (next(it12) != list1.end() && next(it22) != list2.end())
   {
      // make limits for the sublists
      while (*it12 < *next(it12))
         ++it12;
      ++it12; // step beyond the sublist
      while (*it22 < *next(it22))
         ++it22;
      ++it22; // step beyond the sublist

      // working with the sublists----------------------------------
      // until the end of the one of the sublists
      while (it11 != it12 && it21 != it22)
      {
         // compare, dump into the data, move the iterator
         if (*it11 < *it21)
         {
            data.push_back(*it11);
            ++it11;
         }
         else
         {
            data.push_back(*it21);
            ++it21;
         }
      }
      // this is the end of the one of the sublists
      // dumpt the rest of the elements into the data
      while (it11 != it12)
      {
         data.push_back(*it11);
         ++it11;
      }
      while (it21 != it22)
```

```
      {
         data.push_back(*it21);
         ++it21;
      }
      // the ends of both sublists, everything has been dumped into the data
      // both iterators stepped beyond the sublists
      // and are pointing at the first elements or at the end of one of the sublists
   }

   // at this point one of the iterators is pointing at the last element of the sublist
   // dump this element into the data and dump the rest elements from another sublist
   if (next(it12) == list1.end())
   {
      data.push_back(*it11);
      while (it21 != list2.end())
      {
         data.push_back(*it21);
         ++it21;
      }
   }
   else if (next(it22) == list2.end())
   {
      data.push_back(*it21);
      while (it11 != list1.end())
      {
         data.push_back(*it11);
         ++it11;
      }
   }
 */
}
/*************************************************************************
* This function sorts a linked list using a Natural Merge Sort.
* Input:  data -  linked list of data to be sorted.
* Output: data -  sorted linked list
*************************************************************************/
template<class T>
void mergeSort(list<T> & data)
{
   list<T> list1;
   list<T> list2;

   while (!split(data, list1, list2))
   {
      data.clear();
      merge(data, list1, list2);
      list1.clear();
      list2.clear();
   }
}
```

week11.cpp

```
/*********************************************************************
* Program:
*    Week 11, Sorting
*    Brother Ercanbrack, CS 235
* Author:
*    Brother Ercanbrack

* Summary:
*   This is a driver program for showing the heap and merge sorts
*********************************************************************/

#include <sstream>
#include <fstream>
#include <vector>
#include <iostream>        // for CIN and COUT
#include <cstring>         // for strcmp
#include <iomanip>         // for SETW
#include "heap.h"
#include "merge.h"

using namespace std;
```

```
/*********************************************************************
 * MAIN
 * Get the sort name and filename from the command line.
 * call the appropriate sort to sort the data contained in the file.
 *********************************************************************/
int main(int argc, const char* argv[])
{

   string programName;
   string sortName;
   string fileName;

   if (argc < 3)
   {
      cout << "Usage: programName sortName fileName" << endl;
     cin >> programName;
     cin >> sortName;
     cin >> fileName;
   }
   // I changed this else to make it work from the mobaXtern editor
   //else
   //{
      if (sortName == "heap"/*strcmp(argv[1], "heap") == 0*/)
      {

         // read the file into a vector
        vector <int> myVector;
        int data;

        ifstream fin(fileName.c_str());
        if (fin.fail())
        {
           cout << "Could not read the file " << fileName << endl;
        }

        myVector.push_back(-11);
        while (fin >> data)
        {
           myVector.push_back(data);
        }

        fin.close();

         // call your heapsort passing the vector as a parameter
        heapSort(myVector);

         // output the sorted vector.
        for (vector<int>::iterator it = ++myVector.begin(); it != myVector.end(); it++)
           cout << setw(3) << *it;
        cout << endl;
      }
      else if (sortName == "merge"/*strcmp(argv[1], "merge") == 0*/)
      {

         // read the file into a linked list
        list <int> myList;
        int data;

        ifstream fin(fileName.c_str());
        if (fin.fail())
        {
           cout << "Could not read the file " << fileName << endl;
        }

        while (fin >> data)
        {
           myList.push_back(data);
        }

        fin.close();
         // call your merge sort
        mergeSort(myList);

         // output the sorted linked list
        for (list<int>::iterator it = myList.begin(); it != myList.end(); it++)
           cout << setw(3) << *it;
      }
      else
```

```
    {
        cout << "\nInvalid sort name - must be 'heap' or 'merge'" << endl;
    }
//}

return 0;
}
```

vas14001@byui.edu