makefile

```
################################################################
# Program:
#     Week 12, Hash
#     Brother Ercanbrack, CS235
# Author:
#     Yurii Vasiuk
# Summary:
#     The abstrack hash class, the non-

abstrack hash class that inherits from the abstrack one, their use in the spell-checker
################################################################

################################################################
# The main rule
################################################################
a.out: week12.o spellCheck.o
       g++ -o a.out week12.o spellCheck.o -g
       tar -cf week12.tar *.h *.cpp makefile

################################################################
# The individual components
#      week12.o     : the driver program
#      spellCheck.o   : the spell-check program and driver
################################################################
week12.o: hash.h week12.cpp ##list.h
       g++ -c week12.cpp -g

spellCheck.o: hash.h spellCheck.h spellCheck.cpp
       g++ -c spellCheck.cpp -g
```

97/100

Commented [ES1]: Style/Elegance – error handling
Memory leak!

hash.h

```
#include <list>
#include <string>

using namespace std;
/*****************************************
* HASH
* The parent template hash class
*****************************************/
template <typename T>
class Hash
{
public:
   // the constructors
   Hash() : numBuckets(0), numElements(0), buckets(NULL) {}
   Hash(int numBuckets) throw (const char *) ;
   Hash(const Hash & rhs) throw (const char *) ;

   // destructor
   ~Hash()
   {
      //if (numBuckets != 0)
         //delete[] buckets;
   }

   // is the hash empty?
   bool empty() const
   {
      if (numElements == 0)
         return true;
      else
         return false;
   }

   // return the array's capacity
   int capacity() const { return numBuckets; }
```

Commented [ES2]: This should not be commented out.
You need it to not have memory leaks.

```
   // how many elements does the hash have?
   int size() const { return numElements; }

   // clear the content of the hash
   void clear();

   // is the item in the hash?
   bool find(T item) const;

   // insert the item into the hash
   void insert(T item);

   // pure vitual hash function
   virtual int hash(T & value) const = 0;

private:
   int numBuckets;
   int numElements;
   list<T> * buckets;
};

/***************************************
* HASH : NON-DEFAULT CONSTRUCTOR
* Allocate the array of the passed number of list, assign the variables
***************************************/
template <typename T>
Hash<T>::Hash(int numBuckets) throw (const char *)
{
   this->numBuckets = numBuckets;
   numElements = 0;

   try
   {
      buckets = new list<T> [this->numBuckets];
   }
   catch (bad_alloc)
   {
      cout << "ERROR: Unable to allocate memory for the hash.";
   }
}

/***************************************
* HASH : COPY CONSTRUCTOR
* Make a new hash, the same with the passed one
***************************************/
template <typename T>
Hash<T>::Hash(const Hash & rhs) throw (const char *)
{
   // copy the empty hash with no capacity
   if ((rhs.empty() == true) && (rhs.capacity() == 0))
   {
      this->numBuckets = 0;
      this->numElements = 0;
      buckets = NULL;
   }
   // copy the empty hash with some capacity
   else if (rhs.empty() == true)
   {
      this->numBuckets = rhs.numBuckets;
      this->numElements = 0;

      try
      {
         buckets = new list<T>[numBuckets];
      }
      catch (bad_alloc)
      {
         cout << "ERROR: Unable to allocate memory for the hash.";
      }
   }
   // copy the hash with some elements
   else
   {
      this->numBuckets = rhs.numBuckets;
      this->numElements = rhs.numElements;

      try
      {
```

```
         buckets = new list<T>[numBuckets];
      }
      catch (bad_alloc)
      {
         cout << "ERROR: Unable to allocate memory for the hash.";
      }

      for (int i = 0; i < rhs.numBuckets; i++)
      {
         if (!rhs.buckets[i].empty())
         {
            this->buckets = rhs.buckets;
         }
      }
   }
}

/*****************************************
* HASH : CLEAR
* Clear the content of the hash
*****************************************/
template <typename T>
void Hash<T>::clear()
{
   if (!empty())
   {
      for (int i = 0; i < numBuckets; i++)
      {
         if (!buckets[i].empty())
         {
            buckets[i].clear();
         }
      }
   }
}

/*****************************************
* HASH : FIND
* Is the element in the hash?
*****************************************/
template <typename T>
bool Hash<T>::find(T element) const
{
   if (!empty())
   {
      int index = hash(element);
      for (typename list<T>::iterator it = buckets[index].begin(); it != buckets[index].end(); it++)
      {
         if (*it == element)
            return true;
      }
   }

   return false;
}

/*****************************************
* HASH : INSERT
* Insert the element into the hash?
*****************************************/
template <typename T>
void Hash<T>::insert(T element)
{
   int index = hash(element);

   buckets[index].push_back(element);
   numElements++;
}

/*****************************************
* SHASH
* A hash of strings : It inherits from HASH class
*****************************************/
class SHash : public Hash <string>
{
public:
   SHash(int numBuckets)    throw (const char *) : Hash <string>(numBuckets) {}
   SHash(const SHash & rhs) throw (const char *) : Hash <string>(rhs)        {}
```

Commented [ES3]: This is copying pointer to the array over and over again.

It should be:

this->buckets[i] = rsh.buckets[i];

you need to copy each linked list from the rhs array to the left had side array.

This is probably what caused you destructor to fail because you probably freed the same pointer twice!

```
   // hash function for integers is simply to take the modulous
   int hash(string & word) const
   {
      int index = 0;
      int sumLetters = 0;

      for (string::iterator it = word.begin(); it != word.end(); it++)
      {
         sumLetters += static_cast<int>(*it);
      }
      index = sumLetters % capacity();

      return index;
   }
};
```

spellCheck.h

```
/***********************************************************************
 * Module:
 *    Week 12, Spell Check
 *    Brother Helfrich, CS 235
 * Author:
 *    Br. Helfrich
 * Summary:
 *    This program will implement the spellCheck() function
 ************************************************************************/

#ifndef SPELL_CHECK_H
#define SPELL_CHECK_H

void spellCheck();

#endif // SPELL_CHECK_H
```

spellCheck.cpp

```
/***********************************************************************
 * Module:
 *    Week 12, Spell Check
 *    Brother Helfrich, CS 235
 * Author:
 *    Yurii Vasiuk
 * Summary:
 *    This program will implement the spellCheck() function
 ************************************************************************/

#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include "spellCheck.h"
#include "hash.h"
using namespace std;

/****************************************
 * SPELL CHECK
 * Prompt the user for a file to spell-check
 ****************************************/
void spellCheck()
{
   // read the dictionary into the hash object
   SHash hashedDictionary(232);
  string word;
  string wordLowCase;

  string fileName = "/home/cs235/week12/dictionary.txt";
   ifstream fin(fileName.c_str());

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

   while (fin >> word)
   {
      hashedDictionary.insert(word);
```

Commented [ES4]: If the file won't open you can't just display an error and continue. You must either exit() or throw an exception.

```
   }

   fin.close();
   // the dictionary is hashed now


   // the spell check----------------------------------------------------
   cout << "What file do you want to check? ";
   cin >> fileName;

   // container for the misspelled words
   vector<string> misspelledWords;

   ifstream fin1(fileName.c_str());
   if (fin1.fail())
   {
      cout << "Could not read the file " << fileName << endl;
      return;
   }

   while (fin1 >> word)
   {
    wordLowCase = word;
    transform(wordLowCase.begin(), wordLowCase.end(), wordLowCase.begin(), ::tolower);
      if (!hashedDictionary.find(wordLowCase))
         misspelledWords.push_back(word);
   }

   fin1.close();

   // the misspelled words are in the vector now-----------------------------

   // display the result of the check
   if (misspelledWords.empty())
      cout << "File contains no spelling errors\n";
   else
   {
      cout << "Misspelled: ";
      for (vector<string>::iterator it = misspelledWords.begin(); it != misspelledWords.end(); )
    {
         cout << *it;
      if (++it != misspelledWords.end())
            cout << ", ";
    }
    cout << endl;
   }
}
```

week12.cpp

```
/**********************************************************************
* Program:
*    Week 12, Hash
*    Brother Helfrich, CS 235
* Author:
*    Br. Helfrich
* Summary:
*    This is a driver program to exercise the Hash class.  When you
*    submit your program, this should not be changed in any way.  That being
*    said, you may need to modify this once or twice to get it to work.
**********************************************************************/

#include <iostream>      // for CIN and COUT
#include <string>        // for STRING
#include "hash.h"        // for Hash class which should be in hash.h
#include "spellCheck.h"  // for the spellCheck prototype
using namespace std;


// prototypes for our four test functions
void testSimple();
void testAdd();
void testCopy();
void testQuery();

// To get your program to compile, you might need to comment out a few
// of these. The idea is to help you avoid too many compile errors at once.
// I suggest first commenting out all of these tests, then try to use only
```

```
// TEST1.  Then, when TEST1 works, try TEST2 and so on.
#define TEST1   // for testSimple()
#define TEST2   // for testAdd()
#define TEST3   // for testCopy()
#define TEST4   // for testQuery()

/*****************************************************************
 * MAIN
 * This is just a simple menu to launch a collection of tests
 *****************************************************************/
int main()
{
   // menu
   cout << "Select the test you want to run:\n";
   cout << "\t1. Just create and destroy a hash\n";
   cout << "\t2. The above plus add a few entries\n";
   cout << "\t3. The above plus copy a hash table\n";
   cout << "\t4. The above plus look for the entries\n";
   cout << "\ta. Spell check\n";

   // select
   char choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 'a':
         spellCheck();
         break;
      case '1':
         testSimple();
         cout << "Test 1 complete\n";
         break;
      case '2':
         testAdd();
         cout << "Test 2 complete\n";
         break;
      case '3':
         testCopy();
         cout << "Test 3 complete\n";
         break;
      case '4':
         testQuery();
         cout << "Test 4 complete\n";
         break;
      default:
         cout << "Unrecognized command, exiting...\n";
   }

   return 0;
}

/*************************************
 * I HASH
 * A simple hash of integers
 *************************************/
class IHash : public Hash <int>
{
public:
   IHash(int numBuckets)    throw (const char *) : Hash <int> (numBuckets) {}
   IHash(const IHash & rhs) throw (const char *) : Hash <int> (rhs)        {}

   // hash function for integers is simply to take the modulous
   int hash(int & value) const
   {
      return value % capacity();
   }
};

/*************************************
 * F HASH
 * A hash of Floating Point numbers
 *************************************/
class FHash : public Hash <float>
{
public:
   FHash(int numBuckets, float min, float max) throw (const char *) :
         Hash <float> (numBuckets), min(min), max(max)  {}
   FHash(const FHash & rhs) throw (const char *) :
```

```
        Hash <float> (rhs), min(rhs.min), max(rhs.max) {}

   // hash function for strings will add up all the ASCII values
   int hash(float & value) const
   {
      return (int)((value - min) / (max - min) * capacity()) % capacity();
   }
private:
   float min;
   float max;
};

/*****************************************
 * TEST SIMPLE
 * Very simple test for a Hash: create and destroy
 *****************************************/
void testSimple()
{
#ifdef TEST1
   try
   {
      // Test 1.a: bool Set
      cout << "Create an integer Hash\n";
      IHash h1(5);
      cout << "\tSize:     " << h1.size()                  << endl;
      cout << "\tCapacity: " << h1.capacity()              << endl;
      cout << "\tEmpty?    " << (h1.empty() ? "Yes" : "No") << endl;

      // Test 1.b: float Hash
      cout << "Create a float Hash\n";
      FHash h2(10 /*capacity*/, 1.0 /*min*/, 10.0 /*max*/);
      cout << "\tSize:     " << h2.size()                  << endl;
      cout << "\tCapacity: " << h2.capacity()              << endl;
      cout << "\tEmpty?    " << (h2.empty() ? "Yes" : "No") << endl;

      // Test 1.c: copy the Hash using the copy constructor
      {
         cout << "Create a float Hash using the copy constructor\n";
         FHash h3(h2);
         cout << "\tSize:     " << h3.size()                  << endl;
         cout << "\tCapacity: " << h3.capacity()              << endl;
         cout << "\tEmpty?    " << (h3.empty() ? "Yes" : "No") << endl;
      }

      // Test 1.d: copy the Hash using the assignment operator
      cout << "Create a float Hash using the assignment operator\n";
      FHash h4(10, 1.0, 10.0);
      h4 = h2;
      cout << "\tSize:     " << h4.size()                  << endl;
      cout << "\tCapacity: " << h4.capacity()              << endl;
      cout << "\tEmpty?    " << (h4.empty() ? "Yes" : "No") << endl;
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif //TEST1
}

/*****************************************
 * TEST ADD
 * Add a few elements to our Hash
 *****************************************/
void testAdd()
{
#ifdef TEST2
   try
   {
      // create
      cout << "Create a small integer hash\n";
      IHash h(10);
      cout << "\tSize:     " << h.size()                  << endl;
      cout << "\tCapacity: " << h.capacity()              << endl;
      cout << "\tEmpty?    " << (h.empty() ? "Yes" : "No") << endl;

      // fill
      cout << "Fill with 12 values\n";
      h.insert(213);  // h[0] -->
      h.insert(431);  // h[1] --> 431 --> 991 --> 101 --> 111
```

```
      h.insert(534);  // h[2] --> 452 --> 982
      h.insert(452);  // h[3] --> 213 --> 123
      h.insert(123);  // h[4] --> 534
      h.insert(991);  // h[5] --> 005
      h.insert(982);  // h[6] --> 626
      h.insert(626);  // h[7] -->
      h.insert(408);  // h[8] --> 408
      h.insert(101);  // h[9] -->
      h.insert(111);
      h.insert(005);
      cout << "\tSize:     " << h.size()                  << endl;
      cout << "\tCapacity: " << h.capacity()              << endl;
      cout << "\tEmpty?    " << (h.empty() ? "Yes" : "No") << endl;
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif // TEST2
}

/*****************************************
 * TEST COPY
 * Create a few Hashes and copy them into each other
 *****************************************/
void testCopy()
{
#ifdef TEST3
   try
   {
      // create a hash of capacity 25 but 100 items in it
      cout << "A hash of 25 buckets\n";
      IHash h1(25);
      for (int i = 0; i < 100; i++)
         h1.insert(i);
      cout << "\tEmpty?    " << (h1.empty() ? "yes" : "no") << endl;
      cout << "\tSize:     " << h1.size()                   << endl;
      cout << "\tCapacity: " << h1.capacity()               << endl;

      // use the copy constructor
      cout << "Copy the hash into another\n";
      IHash h2(h1);
      cout << "\tEmpty?    " << (h2.empty() ? "yes" : "no") << endl;
      cout << "\tSize:     " << h2.size()                   << endl;
      cout << "\tCapacity: " << h2.capacity()               << endl;

      // create a hash of 10 items and assign h1 into it
      cout << "Create a hash of 5 buckets\n";
      IHash h3(5);
      cout << "\tEmpty?    " << (h3.empty() ? "yes" : "no") << endl;
      cout << "\tSize:     " << h3.size()                   << endl;
      cout << "\tCapacity: " << h3.capacity()               << endl;

      // copy the large hash into this smaller one
      cout << "Copy the large hash of 25 buckets into the small one of 5\n";
      h3 = h1;         // notice the different number of buckets
      h1.insert(100);  // these should not influence h3 in any way
      h1.insert(200);
      cout << "\tEmpty?    " << (h3.empty() ? "yes" : "no") << endl;
      cout << "\tSize:     " << h3.size()                   << endl;
      cout << "\tCapacity: " << h3.capacity()               << endl;
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif // TEST3
}

/*****************************************
 * TEST QUERY
 * Prompt the user for items to put in the hash
 * and then allow the user to query for items
 *****************************************/
void testQuery()
{
#ifdef TEST4
   try
   {
```

```
      cout << "Test adding and querying numbers (0.0 - 100.0) from the hash:\n";
      cout << "  +5.5   Put 5.5 into the hash\n";
      cout << "  ?5.5   Determine if 5.5 is in the hash\n";
      cout << "  !      Display the size and capacity of the hash\n";
      cout << "  #      Quit\n";

      // create the hash
      FHash h(10, 0.0, 100.0);

      // interact
      char instruction;
      float number;
      do
      {
         cout << "> ";
         cin  >> instruction;
         switch (instruction)
         {
            case '+':
               cin >> number;
               h.insert(number);
               break;
            case '?':
               cin >> number;
               cout << '\t'
                    << (h.find(number) ? "Found!" : "Not found.")
                    << endl;
               break;
            case '!':
               cout << "\tSize:     " << h.size()                  << endl;
               cout << "\tCapacity: " << h.capacity()              << endl;
               cout << "\tEmpty?    " << (h.empty() ? "Yes" : "No") << endl;
               break;
            case '#':
               break;
            default:
               cout << "Invalid command\n";
         }
      }
      while (instruction != '#');
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif // TEST4
}
```

Test Bed Results

```
a.out:

--------------------------------------------------------------------
Starting Test 1

   > Select the test you want to run:
   >    1. Just create and destroy a hash
   >    2. The above plus add a few entries
   >    3. The above plus copy a hash table
   >    4. The above plus look for the entries
   >    a. Spell check
   > > 1
Create, destroy, and copy a Hash
   > Create an integer Hash
   >    Size:     0
   >    Capacity: 5
   >    Empty?    Yes
   > Create a float Hash
   >    Size:     0
   >    Capacity: 10
   >    Empty?    Yes
   > Create a float Hash using the copy constructor
   >    Size:     0
   >    Capacity: 10
   >    Empty?    Yes
   > Create a float Hash using the assignment operator
   >    Size:     0
```

```
   >    Capacity: 10
   >    Empty?    Yes
   > Test 1 complete

Test 1 passed.
---------------------------------------------------------------

---------------------------------------------------------------
Starting Test 2

   > Select the test you want to run:
   >    1. Just create and destroy a hash
   >    2. The above plus add a few entries
   >    3. The above plus copy a hash table
   >    4. The above plus look for the entries
   >    a. Spell check
   > > 2

First create a simple hash
   > Create a small integer hash
   >    Size:     0
   >    Capacity: 10
   >    Empty?    Yes

After filling it, the hash should look like this:
 h[0] -->
 h[1] --> 431 --> 991 --> 101 --> 111
 h[2] --> 452 --> 982
 h[3] --> 213 --> 123
 h[4] --> 534
 h[5] --> 005
 h[6] --> 626
 h[7] -->
 h[8] --> 408
 h[9] -->
   > Fill with 12 values
   >    Size:     12
   >    Capacity: 10
   >    Empty?    No
   > Test 2 complete

Test 2 passed.
---------------------------------------------------------------

---------------------------------------------------------------
Starting Test 3

   > Select the test you want to run:
   >    1. Just create and destroy a hash
   >    2. The above plus add a few entries
   >    3. The above plus copy a hash table
   >    4. The above plus look for the entries
   >    a. Spell check
   > > 3

Create a hash of 25 buckets, each one will have exactly 4 elements in it.
   > A hash of 25 buckets
   >    Empty?    no
   >    Size:     100
   >    Capacity: 25

Using the copy constructor, the new hash should have
the same number of buckets as h1
   > Copy the hash into another
   >    Empty?    no
   >    Size:     100
   >    Capacity: 25

Create an empty hash of 5 buckets
   > Create a hash of 5 buckets
   >    Empty?    yes
   >    Size:     0
   >    Capacity: 5

It should be possible to copy the contents of one hash into another,
even when the two hashes have a different number of buckets.
In this case, we will go from 25 buckets with 4 elements in each bucket,
to 5 buckets with 20 elements in each
   > Copy the large hash of 25 buckets into the small one of 5
```

```
   >    Empty?    no
   >    Size:     100
   >    Capacity: 25
   > Test 3 complete

Test 3 passed.
------------------------------------------------------------


------------------------------------------------------------
Starting Test 4

   > Select the test you want to run:
   >    1. Just create and destroy a hash
   >    2. The above plus add a few entries
   >    3. The above plus copy a hash table
   >    4. The above plus look for the entries
   >    a. Spell check
   > > 4

Create a hash of 10 floating point numbers with a range of 0 through 100
   > Test adding and querying numbers (0.0 - 100.0) from the hash:
   >   +5.5   Put 5.5 into the hash
   >   ?5.5   Determine if 5.5 is in the hash
   >   !      Display the size and capacity of the hash
   >   #      Quit
   > > +5.5
   > > > +55.0
   > > > +7.2
 [0] -> 5.5 -> 7.2
 [1]
 [2]
 [3]
 [4]
 [5] -> 55.0
 [6]
 [7]
 [8]
 [9]
   > > !
   >    Size:     3
   >    Capacity: 10
   >    Empty?    No
   > > +8.3
 [0] -> 5.5 -> 7.2 -> 8.3
 [1]
 [2]
 [3]
 [4]
 [5] -> 55.0
 [6]
 [7]
 [8]
 [9]
   > > !
   >    Size:     4
   >    Capacity: 10
   >    Empty?    No
   > > ?5.5
   >    Found!
   > > ?5.4
   >    Not found.
   > > +14.8
   > > > +25.5
   > > > +83.6
   > > > +99.9
   > > > +0.1
 [0] -> 5.5 -> 7.2 -> 8.3 -> 0.1
 [1] -> 14.8
 [2] -> 25.5
 [3]
 [4]
 [5] -> 55.0
 [6]
 [7]
 [8] -> 83.6
 [9] -> 99.9
   > > !
   >    Size:     9
   >    Capacity: 10
```

```
   >    Empty?    No
   > > ? 99.9
   >    Found!
   > > #
   > Test 4 complete

Test 4 passed.
---------------------------------------------------------------

---------------------------------------------------------------
Starting Test 5

   > Select the test you want to run:
   >    1. Just create and destroy a hash
   >    2. The above plus add a few entries
   >    3. The above plus copy a hash table
   >    4. The above plus look for the entries
   >    a. Spell check
   > > a
   > What file do you want to check? /home/cs235/week12/nephi.txt
   > Misspelled: Nephi, yea

Test 5 passed.
---------------------------------------------------------------

---------------------------------------------------------------
Starting Test 6

   > Select the test you want to run:
   >    1. Just create and destroy a hash
   >    2. The above plus add a few entries
   >    3. The above plus copy a hash table
   >    4. The above plus look for the entries
   >    a. Spell check
   > > a
   > What file do you want to check? /home/cs235/week12/twoCities.txt
   > File contains no spelling errors

Test 6 passed.
---------------------------------------------------------------


===============================================================
Passed all tests with no errors.
===============================================================
```

vas14001@byui.edu