Names: Neha Parmar, Shahed Issa

Course Name: Principles of Software Design

Lab Section: B01

Course Code: ENSF 480

Assignment Number: Lab-2

Submission Date and Time: 19/09/2024

Exercise A:

exBmain.cpp:

```
#include <assert.h>
#include <iostream>
#include "dictionaryList.h"

using namespace std;

DictionaryList dictionary_tests();

void test_copying();

void print(DictionaryList& dl);

void test_finding(DictionaryList& dl);

void test_operator_overloading(DictionaryList& dl);

int main()
{
  DictionaryList dl = dictionary_tests();

  test_copying();

 // Uncomment the call to test_copying when DictionaryList::copy is properly defined
 test_finding(dl);
 test_operator_overloading(dl);

  return 0;
}

DictionaryList dictionary_tests()
{

  DictionaryList dl;

  assert(dl.size() == 0);
  cout << "\nPrinting list just after its creation ...\n";
  print(dl);

  // Insert using new keys.
  dl.insert(8001,"Dilbert");
  dl.insert(8002,"Alice");
  dl.insert(8003,"Wally");
  assert(dl.size() == 3);
  cout << "\nPrinting list after inserting 3 new keys ...\n";
  print(dl);
  dl.remove(8002);
  dl.remove(8001);
  dl.insert(8004,"PointyHair");
  assert(dl.size() == 2);
  cout << "\nPrinting list after removing two keys and inserting PointyHair ...\n";
  print(dl);

  // Insert using existing key.
  dl.insert(8003,"Sam");
  assert(dl.size() == 2);
  cout << "\nPrinting list after changing data for one of the keys ...\n";
  print(dl);

  dl.insert(8001,"Allen");
```

```
  dl.insert(8002,"Peter");
  assert(dl.size() == 4);
  cout << "\nPrinting list after inserting 2 more keys ...\n";
  print(dl);

  cout << "***----Finished dictionary tests---------------------------***\n\n";
  return dl;
}

void test_copying()
{
   DictionaryList one;

  // Copy an empty list.
  DictionaryList two;
  assert(two.size() == 0);

  // Copy a list with three entries and a valid cursor.
  one.insert(319,"Randomness");
  one.insert(315,"Shocks");
  one.insert(335,"ParseErrors");
  one.go_to_first();
  one.step_fwd();

  DictionaryList three(one);

  assert(three.cursor_datum().isEqual("Randomness"));
  one.remove(335);

  cout << "Printing list--keys should be 315, 319\n";
  print(one);

  cout << "Printing list--keys should be 315, 319, 335\n";
  print(three);

  // Assignment operator check.
  one = two = three = three;
  one.remove(319);
  two.remove(315);

  cout << "Printing list--keys should be 315, 335\n";
  print(one);

  cout << "Printing list--keys should be 319, 335\n";
  print(two);

  cout << "Printing list--keys should be 315, 319, 335\n";
  print(three);

  cout << "***----Finished tests of copying----------------------***\n\n";
}

void print(DictionaryList& dl)
{
  if (dl.size() == 0)
    cout << "  List is EMPTY.\n";
  for (dl.go_to_first(); dl.cursor_ok(); dl.step_fwd()) {
    cout << "  " << dl.cursor_key();
    cout << "  " << dl.cursor_datum().c_str() << '\n';
  }
}
```

```
void test_finding(DictionaryList& dl)
{

     // Pretend that a user is trying to look up names.
     cout << "\nLet's look up some names ...\n";

     dl.find(8001);
     if (dl.cursor_ok())
        cout << "  name for 8001 is: " << dl.cursor_datum().c_str() << ".\n";
     else
        cout << "  Sorry, I couldn't find 8001 in the list. \n" ;

     dl.find(8000);
     if (dl.cursor_ok())
        cout << "  name for 8000 is: " << dl.cursor_datum().c_str() << ".\n";
     else
        cout << "  Sorry, I couldn't find 8000 in the list. \n" ;

     dl.find(8002);
     if (dl.cursor_ok())
        cout << "  name for 8002 is: " << dl.cursor_datum().c_str() << ".\n";
     else
        cout << "  Sorry, I couldn't find 8002 in the list. \n" ;

     dl.find(8004);
     if (dl.cursor_ok())
        cout << "  name for 8004 is: " << dl.cursor_datum().c_str() << ".\n";
     else
        cout << "  Sorry, I couldn't find 8004 in the list. \n" ;

    cout << "***----Finished tests of finding -------------------------***\n\n";
}
#if 1
void test_operator_overloading(DictionaryList& dl)
{

    DictionaryList dl2 = dl;
    dl.go_to_first();
    dl.step_fwd();
    dl2.go_to_first();

    cout << "\nTestig a few comparison and insertion operators." << endl;

    // Needs to overload >= and << (insertion operator) in class Mystring
    if(dl.cursor_datum() >= (dl2.cursor_datum()))
       cout << endl << dl.cursor_datum() << " is greater than or equal " <<
dl2.cursor_datum();
    else
       cout << endl << dl2.cursor_datum() << " is greater than " <<
dl.cursor_datum();

    // Needs to overload <= for Mystring
    if(dl.cursor_datum() <= (dl2.cursor_datum()))
        cout << dl.cursor_datum() << " is less than or equal" << dl2.cursor_datum();
    else
        cout << endl << dl2.cursor_datum() << " is less than " << dl.cursor_datum();

    if(dl.cursor_datum() != (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is not equal to " <<
dl2.cursor_datum();
    else
        cout << endl << dl2.cursor_datum() << " is equal to " << dl.cursor_datum();
```

```
    if(dl.cursor_datum() > (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is greater than " <<
dl2.cursor_datum();
    else
        cout << endl << dl.cursor_datum() << " is not greater than " <<
dl2.cursor_datum();

    if(dl.cursor_datum() < (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is less than " << dl2.cursor_datum();
    else
        cout << endl << dl.cursor_datum() << " is not less than " <<
dl2.cursor_datum();
    if(dl.cursor_datum() == (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is equal to " << dl2.cursor_datum();
    else
        cout << endl << dl.cursor_datum() << " is not equal to " <<
dl2.cursor_datum();
   cout << endl << "\nUsing square bracket [] to access elements of Mystring
objects. ";

    char c = dl.cursor_datum()[1];
    cout << endl << "The socond element of "  << dl.cursor_datum() << " is: " << c;

    dl.cursor_datum()[1] = 'o';
    c = dl.cursor_datum()[1];
    cout << endl << "The socond element of "  << dl.cursor_datum() << " is: " << c;

    cout << endl << "\nUsing << to display key/datum pairs in a Dictionary list:
\n";
    /* The following line is expected to display the content of the linked list
     * dl2 -- key/datum pairs. It should display:
     *   8001  Allen
     *   8002  Peter
     *   8003  Sam
     *   8004  PointyHair
     */
    cout << dl2;

    cout << endl << "\nUsing [] to display the datum only: \n";
    /* The following line is expected to display the content of the linked list
     * dl2 -- datum. It should display:
     *   Allen
     *   Peter
     *   Sam
     *   PointyHair
     */

    for(int i =0; i < dl2.size(); i++)
        cout << dl2[i] << endl;

    cout << endl << "\nUsing [] to display sequence of charaters in a datum: \n";
    /* The following line is expected to display the characters in the first node
     * of the dictionary. It should display:
     *   A
     *   l
     *   l
     *   e
     *   n
     */
    cout << dl2[0][0] << endl;
    cout << dl2[0][1] << endl;
```

```
    cout << dl2[0][2] << endl;
    cout << dl2[0][3] << endl;
    cout << dl2[0][4] << endl;

    cout << "\n\n***----Finished tests for overloading operators ----------***\n\n";
}
#endif
```

mystring_B.cpp

```
#include "mystring_B.h"
#include <string.h>
#include <iostream>
using namespace std;

ostream &operator<<(ostream &out, const Mystring &c) {
       out << c.charsM;
       return out;
}

bool Mystring::operator>=(const Mystring &rhs) const{
  return (strcmp(charsM, rhs.charsM) >= 0);
}

bool Mystring::operator<=(const Mystring &rhs) const {
  return (strcmp(charsM, rhs.charsM) <= 0);
}

bool Mystring::operator<(const Mystring &rhs) const {
  return (strcmp(charsM, rhs.charsM) < 0);
}

bool Mystring::operator>(const Mystring &rhs) const{
  return (strcmp(charsM, rhs.charsM) > 0);
}

bool Mystring::operator!=(const Mystring &rhs) const {
  return (strcmp(charsM, rhs.charsM) != 0);
}

bool Mystring::operator==(const Mystring &rhs) const {
  return (strcmp(charsM, rhs.charsM) == 0);
}

char &Mystring::operator[](int index) const{
  if(index < 0 || index >= length()){
    cerr << "\nERROR: get_char: the position is out of boundary." ;
  }

  return charsM[index];
}

Mystring::Mystring()
{
  charsM = new char[1];

  // make sure memory is allocated.
  memory_check(charsM);
  charsM[0] = '\0';
```

```
  lengthM = 0;
}

Mystring::Mystring(const char *s)
  : lengthM(strlen(s))
{
  charsM = new char[lengthM + 1];

 // make sure memory is allocated.
  memory_check(charsM);

  strcpy(charsM, s);
}

Mystring::Mystring(int n)
  : lengthM(0), charsM(new char[n])
{
  // make sure memory is allocated.
  memory_check(charsM);
  charsM[0] = '\0';
}

Mystring::Mystring(const Mystring& source):
  lengthM(source.lengthM), charsM(new char[source.lengthM+1])
{
  memory_check(charsM);
  strcpy (charsM, source.charsM);
}

Mystring::~Mystring()
{
  delete [] charsM;
}

int Mystring::length() const
{
  return lengthM;
}

char Mystring::get_char(int pos) const
{
  if(pos < 0 && pos >= length()){
    cerr << "\nERROR: get_char: the position is out of boundary." ;
  }

  return charsM[pos];
}

const char * Mystring::c_str() const
{
  return charsM;
}

void Mystring::set_char(int pos, char c)
{
  if(pos < 0 && pos >= length()){
    cerr << "\nset_char: the position is out of boundary."
        << " Nothing was changed.";
    return;
  }

  if (c != '\0'){
```

```
    cerr << "\nset_char: char c is empty."
        << " Nothing was changed.";
    return;
  }

  charsM[pos] = c;
}

Mystring& Mystring::operator=(const Mystring& S)
{
  if(this == &S)
    return *this;
  delete [] charsM;
  lengthM = (int)strlen(S.charsM);
  charsM = new char [lengthM+1];
  memory_check(charsM);
  strcpy(charsM,S.charsM);

  return *this;
}

Mystring& Mystring::append(const Mystring& other)
{
  char *tmp = new char [lengthM + other.lengthM + 1];
  memory_check(tmp);
  lengthM+=other.lengthM;
  strcpy(tmp, charsM);
  strcat(tmp, other.charsM);
  delete []charsM;
  charsM = tmp;

  return *this;
}

 void Mystring::set_str(char* s)
{
    delete []charsM;
    lengthM = (int)strlen(s);
    charsM=new char[lengthM+1];
    memory_check(charsM);

    strcpy(charsM, s);
}

int Mystring::isEqual (const Mystring& s)const
{
  return (strcmp(charsM, s.charsM)== 0);
}


void Mystring::memory_check(char* s)
{
  if(s == 0)
    {
      cerr <<"Memory not available.";
      exit(1);
    }
}
```

mystring_B.h

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

#ifndef MYSTRING_H
#define MYSTRING_H

class Mystring {

public:

  // adding overload operators
  friend ostream &operator<<(ostream &out, const Mystring &c);
  bool operator>=(const Mystring &rhs) const;
  bool operator<=(const Mystring &rhs) const;
  bool operator<(const Mystring &rhs) const;
  bool operator>(const Mystring &rhs) const;
  bool operator!=(const Mystring &rhs) const;
  bool operator==(const Mystring &rhs) const;
  char &operator[](int index) const;

  Mystring();
  // PROMISES: Empty string object is created.

  Mystring(int n);
  // PROMISES: Creates an empty string with a total capacity of n.
  //           In other words, dynamically allocates n elements for
  //           charsM,sets the lengthM to zero, and fills the first
  //           element of charsM with '\0'.

  Mystring(const char *s);
  // REQUIRES: s points to first char of a built-in string.
  // REQUIRES: Mystring object is created by copying chars from s.

  ~Mystring(); // destructor

  Mystring(const Mystring& source); // copy constructor

  Mystring& operator =(const Mystring& rhs); // assignment operator
  // REQUIRES: rhs is reference to a Mystring as a source
  // PROMISES: to make this-object (object that this is pointing to, as  a copy
  //           of rhs.

  int length() const;
  // PROMISES: Return value is number of chars in charsM.

  char get_char(int pos) const;
  // REQUIRES: pos >= 0 && pos < length()
  // PROMISES:
  // Return value is char at position pos.
  // (The first char in the charsM is at position 0.)

  const char * c_str() const;
  // PROMISES:
  //   Return value points to first char in built-in string
  //   containing the chars of the string object.

  void set_char(int pos, char c);
  // REQUIRES: pos >= 0 && pos < length(), c != '\0'
  // PROMISES: Character at position pos is set equal to c.

  Mystring& append(const Mystring& other);
```

```
  // PROMISES: extends the size of charsM to allow concatenate other.charsM to
  //           to the end of charsM. For example if charsM points to "ABC", and
  //          other.charsM points to XYZ, extends charsM to "ABCXYZ".
  //

  void set_str(char* s);
  // REQUIRES: s is a valid C++ string of characters (a built-in string)
  // PROMISES:copys s into charsM, if the length of s is less than or equal lengthM.
  //          Othrewise, extends the size of the charsM to s.lengthM+1, and copies
  //          s into the charsM.



  int isEqual (const Mystring& s)const;
  // REQUIRES: s refers to an object of class Mystring
  // PROMISES: retruns true if charsM equal s.charsM.


 private:

  int lengthM; // the string length - number of characters excluding \0
  char* charsM; // a pointer to the beginning of an array of characters, allocated
dynamically.
  void memory_check(char* s);
  // PROMISES: if s points to NULL terminates the program.
};
#endif
```

dictionaryList.cpp

```
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "dictionaryList.h"
#include "mystring_B.h"

using namespace std;

Node::Node(const Key& keyA, const Datum& datumA, Node *nextA)
  : keyM(keyA), datumM(datumA), nextM(nextA)
{
}

DictionaryList::DictionaryList()
  : sizeM(0), headM(0), cursorM(0)
{
}

DictionaryList::DictionaryList(const DictionaryList& source)
{
  copy(source);
}

DictionaryList& DictionaryList::operator =(const DictionaryList& rhs)
{
  if (this != &rhs) {
    destroy();
    copy(rhs);
  }
  return *this;
```

```
}

DictionaryList::~DictionaryList()
{
  destroy();
}

int DictionaryList::size() const
{
  return sizeM;
}

int DictionaryList::cursor_ok() const
{
  return cursorM != 0;
}

const Key& DictionaryList::cursor_key() const
{
  assert(cursor_ok());
  return cursorM->keyM;
}

const Datum& DictionaryList::cursor_datum() const
{
  assert(cursor_ok());
  return cursorM->datumM;
}

void DictionaryList::insert(const int& keyA, const Mystring& datumA)
{
  // Add new node at head?
  if (headM == 0 || keyA < headM->keyM) {
    headM = new Node(keyA, datumA, headM);
    sizeM++;
  }

  // Overwrite datum at head?
  else if (keyA == headM->keyM)
    headM->datumM = datumA;

  // Have to search ...
  else {

    //POINT ONE

    // if key is found in list, just overwrite data;
    for (Node *p = headM; p !=0; p = p->nextM)
              {
                     if(keyA == p->keyM)
                     {
                            p->datumM = datumA;
                            return;
                     }
              }

    //OK, find place to insert new node ...
    Node *p = headM ->nextM;
    Node *prev = headM;

    while(p !=0 && keyA >p->keyM)
              {
```

```
                    prev = p;
                    p = p->nextM;
                }

    prev->nextM = new Node(keyA, datumA, p);
    sizeM++;
  }
  cursorM = NULL;

}

void DictionaryList::remove(const int& keyA)
{
    if (headM == 0 || keyA < headM -> keyM)
        return;

    Node *doomed_node = 0;

    if (keyA == headM-> keyM) {
        doomed_node = headM;
        headM = headM->nextM;

        // POINT TWO
    }
    else {
        Node *before = headM;
        Node *maybe_doomed = headM->nextM;
        while(maybe_doomed != 0 && keyA > maybe_doomed-> keyM) {
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->nextM;
        }

        if (maybe_doomed != 0 && maybe_doomed->keyM == keyA) {
            doomed_node = maybe_doomed;
            before->nextM = maybe_doomed->nextM;
        }


    }
    if(doomed_node == cursorM)
        cursorM = 0;

    delete doomed_node;           // Does nothing if doomed_node == 0.
    sizeM--;
}

void DictionaryList::go_to_first()
{
    cursorM = headM;
}

void DictionaryList::step_fwd()
{
    assert(cursor_ok());
    cursorM = cursorM->nextM;
}

void DictionaryList::make_empty()
{
    destroy();
    sizeM = 0;
    cursorM = 0;
```

```
}


// The following function are supposed to be completed by the stuents, as part
// of the exercise B part II. the given fucntion are in fact place-holders for
// find, destroy and copy, in order to allow successful linking when you're
// testing insert and remove. Replace them with the definitions that work.

void DictionaryList::find(const Key &keyA)
{
  if (headM == 0)
  {
    cursorM = 0;
    return;
  }

  else if (headM->keyM == keyA)
  {
    cursorM = headM;
    return;
  }

  else
  {
    Node *prev = headM;
    Node *next = headM->nextM;

    while (next != 0 && keyA >= prev->keyM)
    {

      if (prev->keyM == keyA)
      {
        cursorM = prev;
        return;
      }

      prev = next;
      next = prev->nextM;
    }

    if (prev->keyM == keyA)
    {
      cursorM = prev;
      return;
    }
  }

  cursorM = 0;
  return;
}

void DictionaryList::destroy()
{
  if (headM == 0)
  {
    return;
  }

  cursorM = 0;
  sizeM = 0;

  Node *prev = headM;
```

```
  Node *next = prev->nextM;

  do
  {

    delete prev;

    prev = next;
    next = prev->nextM;
  } while (next != 0);

  delete prev;
  headM = 0;
}

void DictionaryList::copy(const DictionaryList &source)
{

  if (source.headM == 0)
  {
    this->headM = 0;
    this->cursorM = 0;
    this->sizeM = 0;
    return;
  }

  this->cursorM = 0;
  this->sizeM = 0;
  Node *prev = source.headM;
  Node *next = prev->nextM;

  Node *copy = new Node(prev->keyM, prev->datumM, 0);
  Node *head = copy;
  this->sizeM++;

  while (next != 0)
  {

    copy->nextM = new Node(next->keyM, next->datumM, 0);

    if (source.cursorM == prev)
    {
      this->cursorM = copy;
    }

    copy = copy->nextM;
    this->sizeM++;

    prev = next;
    next = prev->nextM;
  }

  copy->nextM = next;
  this->headM = head;
}

ostream &operator<<(ostream &out, const DictionaryList &c) {
       Node *p = c.headM;
       Node *after = p->nextM;

       while (after != 0)
       {
```

```
                out << p->keyM << " " << p->datumM << endl;
                p = after;
                after = p->nextM;
        }

        out << p->keyM << " " << p->datumM << endl;
        return out;
}

Datum &DictionaryList::operator[](int index) const {
        if (index > size())
        {
                cerr << endl
                        << "index is too high" << endl;
        }
        Node *p = headM;

        while (index > 0)
        {
                p = p->nextM;
                index--;
        }

        return p->datumM;
}
```

disctionaryList.h

```
#ifndef DICTIONARY_H
#define DICTIONARY_H
#include <iostream>
using namespace std;

// class DictionaryList: GENERAL CONCEPTS
//
//    key/datum pairs are ordered.  The first pair is the pair with
//    the lowest key, the second pair is the pair with the second
//    lowest key, and so on.  This implies that you must be able to
//    compare two keys with the < operator.
//
//    Each DictionaryList object has a "cursor" that is either attached
//    to a particular key/datum pair or is in an "off-list" state, not
//    attached to any key/datum pair.  If a DictionaryList is empty, the
//    cursor is automatically in the "off-list" state.

#include "mystring_B.h"

// Edit these typedefs to change the key or datum types, if necessary.
typedef int Key;
typedef Mystring Datum;

// THE NODE TYPE
//    In this exercise the node type is a class, that has a ctor.
//    Data members of Node are private, and class DictionaryList
//    is declared as a friend. For details on the friend keyword refer to your
//    lecture notes.

class DictionaryList;

class Node {
  friend class DictionaryList;
```

```
  friend ostream &operator<<(ostream &out, const DictionaryList &c);

private:
  Key keyM;
  Datum datumM;
  Node *nextM;

  // This ctor should be convenient in insert and copy operations.
  Node(const Key& keyA, const Datum& datumA, Node *nextA);
};

class DictionaryList {
public:
  DictionaryList();
  DictionaryList(const DictionaryList& source);
  DictionaryList& operator =(const DictionaryList& rhs);
  ~DictionaryList();

  friend ostream &operator<<(ostream &out, const DictionaryList &c);
  Datum &operator[](int index) const;

  int size() const;
  // PROMISES: Returns number of keys in the table.

  int cursor_ok() const;
  // PROMISES:
  //   Returns 1 if the cursor is attached to a key/datum pair,
  //   and 0 if the cursor is in the off-list state.

  const Key& cursor_key() const;
  // REQUIRES: cursor_ok()
  // PROMISES: Returns key of key/datum pair to which cursor is attached.

  const Datum& cursor_datum() const;
  // REQUIRES: cursor_ok()
  // PROMISES: Returns datum of key/datum pair to which cursor is attached.

  void insert(const Key& keyA, const Datum& datumA);
  // PROMISES:
  //   If keyA matches a key in the table, the datum for that
  //   key is set equal to datumA.
  //   If keyA does not match an existing key, keyA and datumM are
  //   used to create a new key/datum pair in the table.
  //   In either case, the cursor goes to the off-list state.

  void remove(const Key& keyA);
  // PROMISES:
  //   If keyA matches a key in the table, the corresponding
  //   key/datum pair is removed from the table.
  //   If keyA does not match an existing key, the table is unchanged.
  //   In either case, the cursor goes to the off-list state.

  void find(const Key& keyA);
  // PROMISES:
  //   If keyA matches a key in the table, the cursor is attached
  //   to the corresponding key/datum pair.
  //   If keyA does not match an existing key, the cursor is put in
  //   the off-list state.

  void go_to_first();
  // PROMISES: If size() > 0, cursor is moved to the first key/datum pair
  //   in the table.
```

```
  void step_fwd();
  // REQUIRES: cursor_ok()
  // PROMISES:
  //   If cursor is at the last key/datum pair in the list, cursor
  //   goes to the off-list state.
  //   Otherwise the cursor moves forward from one pair to the next.

  void make_empty();
  // PROMISES: size() == 0.

private:
  int sizeM;
  Node *headM;
  Node *cursorM;

  void destroy();
  // Deallocate all nodes, set headM to zero.

  void copy(const DictionaryList& source);
  // Establishes *this as a copy of source.  Cursor of *this will
  // point to the twin of whatever the source's cursor points to.

};

#endif
```

Output:

```
shahe@Shaheds_Laptop ~/ENSF480/Lab2
$ ./exA

Printing list just after its creation ...
  List is EMPTY.

Printing list after inserting 3 new keys ...
  8001  Dilbert
  8002  Alice
  8003  Wally

Printing list after removing two keys and inserting PointyHair ...
  8003  Wally
  8004  PointyHair

Printing list after changing data for one of the keys ...
  8003  Sam
  8004  PointyHair

Printing list after inserting 2 more keys ...
  8001  Allen
  8002  Peter
  8003  Sam
  8004  PointyHair
***----Finished dictionary tests---------------------------***

Printing list--keys should be 315, 319
  315  Shocks
  319  Randomness
Printing list--keys should be 315, 319, 335
  315  Shocks
  319  Randomness
  335  ParseErrors
Printing list--keys should be 315, 335
  315  Shocks
  335  ParseErrors
Printing list--keys should be 319, 335
  319  Randomness
  335  ParseErrors
Printing list--keys should be 315, 319, 335
  315  Shocks
  319  Randomness
  335  ParseErrors
***----Finished tests of copying----------------------***


Testig a few comparison and insertion operators.

Peter is greater than or equal Allen
Allen is less than Peter
Peter is not equal to Allen
Peter is greater than Allen
Peter is not less than Allen
Peter is not equal to Allen

Using square bracket [] to access elements of Mystring objects.
The socond element of Peter is: e
The socond element of Poter is: o
```

```
Using << to display key/datum pairs in a Dictionary list:
8001 Allen
8002 Peter
8003 Sam
8004 PointyHair


Using [] to display the datum only:
Allen
Peter
Sam
PointyHair


Using [] to display sequence of charaters in a datum:
A
l
l
e
n


***----Finished tests for overloading operators ----------***


shahe@Shaheds_Laptop ~/ENSF480/Lab2
$
```

Exercise B:

GraphicsWorld.cpp:

```
#include <iostream>
#include "GraphicsWorld.h"
#include "Point.h"
#include "Square.h"
#include "Rectangle.h"

int main() {
    GraphicsWorld gw;
    gw.run();
    return 0;
}

void GraphicsWorld::run() {
    std::cout << "made by Neha Parmar and Shahed Issa";

    // Change 0 to 1 to test Point
    #if 1
    Point m(6, 8);
    Point n(6, 8);
    n.setx(9);
    std::cout << "\nExpected to display the distance between m and n is: 3";
    std::cout << "\nThe distance between m and n is: " << m.distance(n);
    std::cout << "\nExpected second version of the distance function also print: 3";
    std::cout << "\nThe distance between m and n is again: " << Point::distance(m,
n);
    #endif // end of block to test Point

    // Change 0 to 1 to test Square
    #if 1
    std::cout << "\n\nTesting Functions in class Square:" << std::endl;
    Point p1(5, 7);  // Create a Point object for Square constructor
    Square s(p1, "SQUARE - S", 12);
    s.display();
    #endif // end of block to test Square

    // Change 0 to 1 to test Rectangle
    #if 1
    std::cout << "\nTesting Functions in class Rectangle:";
    Point p2(5, 7);  // Create Point object for Rectangle constructor
    Rectangle a(p2, "RECTANGLE A", 12, 15);
    a.display();

    Point p3(16, 7);  // Create Point object for another Rectangle
    Rectangle b(p3, "RECTANGLE B", 8, 9);
    b.display();

    double d = a.distance(b);
    std::cout << "\nDistance between rectangle a and b is: " << d << std::endl;

    Rectangle rec1 = a;
    rec1.display();

    std::cout << "\nTesting assignment operator in class Rectangle:" << std::endl;
    Point p4(3, 4);  // Create Point object for another Rectangle
    Rectangle rec2(p4, "RECTANGLE rec2", 11, 7);
    rec2.display();
    rec2 = a;
```

```
    a.set_side_b(200);
    a.set_side_a(100);

    std::cout << "\nExpected to display the following values for object rec2: " <<
std::endl;
    std::cout << "Rectangle Name: RECTANGLE A\n"
              << "X-coordinate: 5\n"
              << "Y-coordinate: 7\n"
              << "Side a: 12\n"
              << "Side b: 15\n"
              << "Area: 180\n"
              << "Perimeter: 54\n";
    std::cout << "\nIf it doesn't, there is a problem with your assignment
operator.\n" << std::endl;
    rec2.display();

    std::cout << "\nTesting copy constructor in class Rectangle:" << std::endl;
    Rectangle rec3(a);
    rec3.display();

    a.set_side_b(300);
    a.set_side_a(400);

    std::cout << "\nExpected to display the following values for object rec3: " <<
std::endl;
    std::cout << "Rectangle Name: RECTANGLE A\n"
              << "X-coordinate: 5\n"
              << "Y-coordinate: 7\n"
              << "Side a: 100\n"
              << "Side b: 200\n"
              << "Area: 20000\n"
              << "Perimeter: 600\n";
    std::cout << "\nIf it doesn't, there is a problem with your copy constructor.\n"
<< std::endl;
    rec3.display();
    #endif // end of block to test Rectangle

    // Change 0 to 1 to test using array of pointers and polymorphism
    #if 1
    std::cout << "\nTesting array of pointers and polymorphism:" << std::endl;
    Shape* sh[4];
    sh[0] = &s;
    sh[1] = &b;
    sh[2] = &rec1;
    sh[3] = &rec3;

    sh[0]->display();
    sh[1]->display();
    sh[2]->display();
    sh[3]->display();
    #endif // end of block to test array of pointers and polymorphism
}
```

GraphicsWorld.h:

```
#ifndef GRAPHICS_WORLD_H
#define GRAPHICS_WORLD_H
```

```
#include "Point.h"
#include "Square.h"
#include "Rectangle.h"
#include "Shape.h"

class GraphicsWorld {
public:
    void run();
};

#endif
```

Point.cpp:

```
#include "Point.h"
#include <iostream>
#include <cmath>
using namespace std;

int Point::count = 1001;

Point::Point(double xCoord, double yCoord) : x(xCoord), y(yCoord) {
    id = count++;
}

double Point::getx() const {
    return x;
}

double Point::gety() const {
    return y;
}

void Point::setx(double xCoord) {
    x = xCoord;
}

void Point::sety(double yCoord) {
    y = yCoord;
}

void Point::display() const {
    cout << "X-coordinate: " << x << endl;
    cout << "Y-coordinate: " << y << endl;
}

int Point::counter() {
    return count - 1001;
}

double Point::distance(const Point& other) const {
    return sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
}

double Point::distance(const Point& p1, const Point& p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}
```

Point.h:

```
#ifndef POINT_H
#define POINT_H

class Point {
private:
    static int count;
    int id;
    double x, y;

public:
    Point(double xCoord, double yCoord);

    void display() const;

    static int counter();

    double distance(const Point& other) const;

    static double distance(const Point& p1, const Point& p2);

    // Getters
    double getx() const;
    double gety() const;

    // Setters
    void setx(double xCoord);
    void sety(double yCoord);
};

#endif
```

Rectangle.cpp:

```
#include "Rectangle.h"
#include <iostream>
#include <cstring>
#include <cmath>
using namespace std;

Rectangle::Rectangle(const Point& p, const char* name, double side_a, double side_b)
    : Square(p, name, side_a), side_b(side_b) {}

Rectangle::Rectangle(const Rectangle& other) : Square(other), side_b(other.side_b) {
}

Rectangle& Rectangle::operator=(const Rectangle& rhs) {
    if (this != &rhs) {
        Square::operator=(rhs);
        side_b = rhs.side_b;
    }
    return *this;
}

Rectangle::~Rectangle() {}

double Rectangle::area() const {
    return get_side_a() * side_b;
}

double Rectangle::perimeter() const {
```

```
    return 2 * (get_side_a() + side_b);
}

double Rectangle::get_side_b() const {
    return side_b;
}

void Rectangle::set_side_b(double side) {
    side_b = side;
}

void Rectangle::display() const {
    cout << "Rectangle Name: " << getName() << endl;
    cout << "X-coordinate: " << getOrigin().getx() << endl;
    cout << "Y-coordinate: " << getOrigin().gety() << endl;
    cout << "Width (Side a): " << get_side_a() << endl;
    cout << "Height (Side b): " << side_b << endl;
    cout << "Area: " << area() << endl;
    cout << "Perimeter: " << perimeter() << endl;
}
```

Rectangle.h:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "Square.h"

class Rectangle : public Square {
private:
    double side_b;

public:
    Rectangle(const Point& p, const char* name, double side_a, double
side_b);
    Rectangle(const Rectangle& other);
    Rectangle& operator=(const Rectangle& rhs);

    ~Rectangle();

    double area() const;

    double perimeter() const;

    double get_side_b() const;
    void set_side_b(double side);

    void display() const;
};

#endif // RECTANGLE_H
```

Shape.cpp:

```
#include "Shape.h"
#include <iostream>
#include <cstring>
```

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

//contructor
Shape::Shape(const Point& p, const char* name) : origin(p) {
    shapeName = new char[strlen(name) + 1];
    strcpy(shapeName, name);
}

//Copy contructor
Shape::Shape(const Shape& other) : origin(other.origin){
    shapeName = new char[strlen(other.shapeName)+1];
    strcpy(shapeName, other.shapeName);
}

//assignment operator
Shape& Shape::operator=(const Shape& rhs){
    if (this != &rhs){
        origin = rhs.origin;
        delete[] shapeName;

        shapeName = new char[strlen(rhs.shapeName)+1];
        strcpy(shapeName, rhs.shapeName);

    }
}

Shape::~Shape() {
    delete[] shapeName;
}


const Point& Shape::getOrigin() const {
    return origin;
}

const char* Shape::getName() const {
    return shapeName;
}

void Shape::display() const {
    cout << "Shape Name: " << shapeName << endl;
    cout << "X-coordinate: " << origin.getx() << endl;
    cout << "Y-coordinate: " << origin.gety() << endl;
}


double Shape::distance(const Shape& other) const {
    return Point::distance(this->origin, other.origin);
}


double Shape::distance(const Shape& shape1, const Shape& shape2) {
    return Point::distance(shape1.origin, shape2.origin);

}

void Shape::move(double dx, double dy) {
    origin.setx(origin.getx() + dx);
    origin.sety(origin.gety() + dy);
}
```

Shape.h:

```
#ifndef SHAPE_H
#define SHAPE_H

#include "Point.h"

class Shape {
private:
    Point origin;
    char* shapeName;

public:
    Shape(const Point& p, const char* name);
    Shape(const Shape& other);
    Shape& operator=(const Shape& rhs);

    ~Shape();

    const Point& getOrigin() const;

    const char* getName() const;

    void display() const;

    double distance(const Shape& other) const;
    static double distance(const Shape& shape1, const Shape& shape2);

    void move(double dx, double dy);
};

#endif
```

Square.cpp:

```
#include "Square.h"
#include <iostream>
#include <cstring>
#include <cmath>
using namespace std;

//constructor
Square::Square(const Point& p, const char* name, double side_a)
    : Shape(p, name), side_a(side_a) {}

//copy constrcutor
Square::Square(const Square& other) : Shape(other), side_a(other.side_a) {

}

//assignment operator
Square& Square::operator=(const Square& rhs) {
    if (this != &rhs) {
        Shape::operator=(rhs);
        side_a = rhs.side_a;
    }
    return *this;
}
```

```
//destructor:
Square::~Square() {
}

double Square::area() const {
    return side_a * side_a;
}

double Square::perimeter() const {
    return side_a * 4;
}

double Square::get_side_a() const {
    return side_a;
}

void Square::set_side_a(double side) {
    side_a = side;
}

void Square::display() const {
    cout << "Square Name: " << getName() << endl;
    cout << "X-coordinate: " << getOrigin().getx() << endl;
    cout << "Y-coordinate: " << getOrigin().gety() << endl;
    cout << "Side a: " << side_a << endl;
    cout << "Area: " << area() << endl;
    cout << "Perimeter: " << perimeter() << endl;
}
```

Square.h:

```
#ifndef SQUARE_H
#define SQUARE_H
#include "Shape.h"
#include "Point.h"

class Square : public Shape {
protected:
    double side_a;

public:
    Square(const Point& p, const char* name, double side_a);
    Square(const Square& other);
    Square& operator=(const Square& rhs);

    ~Square();

    double area() const;

    double perimeter() const;

    double get_side_a() const;
    void set_side_a(double side);

    void display() const;
};
#endif
```

Output:

```
nehap@Neha-Laptop MINGW64 ~/Desktop/ENSF-480/LAB2/ExerciseB (main)
$  g++ -mconsole -o GraphicsWorld GraphicsWorld.cpp Point.cpp Square.cpp Rectangle.cpp Shape.cpp

nehap@Neha-Laptop MINGW64 ~/Desktop/ENSF-480/LAB2/ExerciseB (main)
$ ./GraphicsWorld
made by Neha Parmar and Shahed Issa
Expected to display the distance between m and n is: 3
The distance between m and n is: 3
Expected second version of the distance function also print: 3
The distance between m and n is again: 3

Testing Functions in class Square:
Square Name: SQUARE - S
X-coordinate: 5
Y-coordinate: 7
Side a: 12
Area: 144
Perimeter: 48

Testing Functions in class Rectangle:Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Width (Side a): 12
Height (Side b): 15
Area: 180
Perimeter: 54
Rectangle Name: RECTANGLE B
X-coordinate: 16
Y-coordinate: 7
Width (Side a): 8
Height (Side b): 9
Area: 72
Perimeter: 34

Distance between rectangle a and b is: 11
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Width (Side a): 12
Height (Side b): 15
Area: 180
Perimeter: 54
```

```
Testing assignment operator in class Rectangle:
Rectangle Name: RECTANGLE rec2
X-coordinate: 3
Y-coordinate: 4
Width (Side a): 11
Height (Side b): 7
Area: 77
Perimeter: 36

Expected to display the following values for object rec2:
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 12
Side b: 15
Area: 180
Perimeter: 54

If it doesn't, there is a problem with your assignment operator.

Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Width (Side a): 12
Height (Side b): 15
Area: 180
Perimeter: 54

Testing copy constructor in class Rectangle:
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Width (Side a): 100
Height (Side b): 200
Area: 20000
Perimeter: 600
```

```
Expected to display the following values for object rec3:
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 100
Side b: 200
Area: 20000
Perimeter: 600

If it doesn't, there is a problem with your copy constructor.

Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Width (Side a): 100
Height (Side b): 200
Area: 20000
Perimeter: 600

Testing array of pointers and polymorphism:
Shape Name: SQUARE - S
X-coordinate: 5
Y-coordinate: 7
Shape Name: RECTANGLE B
X-coordinate: 16
Y-coordinate: 7
Shape Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Shape Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
```