Lab 02 Report **Completed by: Dominic Choi 30109955**

Nathan Ante 30157706

Exercise A:

Output

```
mackante@MacKante-PC:/mnt/c/Users/MacKante/Desktop/Fall23Resources/ENSF-480-Labs/Lab2/exA$ make
g++ -o myprog.out exBmain.cpp mystring_B.cpp dictionaryList.cpp
mackante@MacKante-PC:/mnt/c/Users/MacKante/Desktop/Fall23Resources/ENSF-480-Labs/Lab2/exA$ ./myprog.out

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---------------------------***
```

```
  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 second 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 ----------***
```

Code:
dictionaryList.h

```
// dictionaryList.h
// ENSF 480 - Lab 2 - Exercise A

#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 Node {
public:
  Key getKey() const;
  Datum getDatum() const;
```

```
  Node* getNextNode() const;
  friend std::ostream& operator<<(std::ostream& os, const Node& rhs);
  string operator[](int i) const;
  Node();
private:
  Key keyM;
  Datum datumM;
  Node *nextM;

  friend class DictionaryList;
  // 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();

  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.

  // Overloaded operators
  friend std::ostream& operator<<(std::ostream& os, const DictionaryList& rhs);
  Node operator[] (int i) const;

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
```

dictionaryList.cpp

```
// lookuptable.cpp

// ENSF 480 - Lab 2 - Exercise A

// Completed by:
// Nathan Ante & Dominic Choi

#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)
      return;

  // Set current node as headM, go through each node and check if keyA ==
current->keyM
  // If keyA == current->keyM, then set cursorM as current
```

```
  // If no matching key, set cursorM as 0 and return
  Node* current = headM;
  while(current != NULL) {
    if (current->keyM ==  keyA) {
      cursorM = current;
      return;
    }
    current = current->nextM;
  }

  // Key is not within list
  cursorM = 0;
  return;

}

void DictionaryList::destroy()
{
  if (this->headM == NULL || this->headM->nextM == NULL) {
    headM = 0;
    return;
  }

  while(this->headM->nextM != NULL) {
    Node* currentLast = this->headM;
    while (currentLast->nextM->nextM != NULL)
    {
      currentLast = currentLast->nextM;
    }
    currentLast->nextM = NULL;
  }
  headM = 0;
}


void DictionaryList::copy(const DictionaryList& source)
{
  this->sizeM = 0;
  this->cursorM = 0;
```

```
  this->headM = 0;

  if (source.headM == 0) {
    return;
  }

  this->headM = new Node(source.headM->keyM, source.headM->datumM, this->headM);
  this->sizeM++;

  Node* current = this->headM;
  Node* currentSource = source.headM->nextM;
  while(currentSource != NULL) {
    current->nextM = new Node(currentSource->keyM, currentSource->datumM, NULL);
    current = current->nextM;
    currentSource = currentSource->nextM;
    this->sizeM++;
  }

  this->cursorM = source.cursorM;
  return;
}

/*--------------------------------------------------------------------------*/
// Necessary functions for overloaded operators

// default constructor for node;
Node::Node() {
  keyM = 0;
  datumM = 0;
  nextM = nullptr;
}

// getters for node variables
Key Node::getKey() const{
  return this->keyM;
}

Datum Node::getDatum() const{
  return this->datumM;
```

```
}

Node* Node::getNextNode() const{
  return this->nextM;
}


/*---------------------------------------------------------------------------*/
/* Overload operators start here */

// overload [] operator for DictionaryList
Node DictionaryList::operator[](int i) const {
  int j = 0;
  Node* current = this->headM;
  while((current != NULL) && (j < i)) {
    current = current->getNextNode();
    j++;
  }

  if (current == NULL) {
    Node* temp = new Node();
    return (*temp);
  } else {
    return *current;
  }
}

// overload [] operator for node
string Node::operator[](int i) const {
  if (i > this->getDatum().length()-1 || i < 0) {
    string str = "Index out of bounds";
    return str;
  } else {
    string str(1,this->getDatum().get_char(i));
    return (str);
  }
}

// overload << operator for Node object
```

```
std::ostream& operator<<(std::ostream& os, const Node& rhs) {
  if (rhs.getKey() == 0 && rhs.getDatum() == 0 && rhs.getNextNode() == nullptr) {
    os << "Index out of bounds";
  } else {
    Datum datumTemp = rhs.getDatum();
    os << datumTemp;
  }
  return os;
}

// overload << operator for DictionaryList object
std::ostream& operator<<(std::ostream& os, const DictionaryList& rhs) {
  Node* current = rhs.headM;
  while (current != NULL) {
    os << current->getKey() << "  " << current->getDatum() << endl;
    current = current->getNextNode();
  }
  return os;
}
```

mystring_B.h

```
/* File: mystring_B.h
 *
 *
 */
// ENSF 480 - Lab 2 - Exercise A
#include <iostream>
#include <string>
using namespace std;

#ifndef MYSTRING_H
#define MYSTRING_H

class Mystring {

public:
  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 isGreaterThan( const Mystring& s)const;
  // REQUIRES: s refers to an object of class Mystring
  // PROMISES: retruns true if charsM is greater than s.charsM.

  int isLessThan (const Mystring& s)const;
  // REQUIRES: s refers to an object of class Mystring
  // PROMISES: retruns true if charsM is less than s.charsM.

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

  int isNotEqual(const Mystring& s)const;
  // REQUIRES: s refers to an object of class Mystring
  // PROMISES: retruns true if charsM is not equal s.charsM.

  // overloaded operators
  friend std::ostream& operator << (std::ostream& os, const Mystring& rhs);
  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 i) const;
 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
```

Mystring_b.cpp

```
/*  mystring_B.cpp
 *
 *
 */
// ENSF 480 - Lab 2 - Exercise A
#include "mystring_B.h"
#include <string.h>
#include <iostream>
using namespace std;

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

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


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

int Mystring::isLessThan (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);
    }
}

/* Overloaded operators */
std::ostream& operator<< (std::ostream& os, const Mystring& S) {
  os << S.c_str();
  return os;
}

// Overloading >= operator
bool Mystring::operator>= (const Mystring& S) const {
  if (this->isGreaterThan(S)==1 || this->isEqual(S)==1) {
    return true;
  } else {
    return false;
  }
}

// Overloading <= operator
bool Mystring::operator<= (const Mystring& S) const {
  if (this->isLessThan(S)==1 || this->isEqual(S)==1) {
    return true;
  } else {
    return false;
  }
}

//// Overloading != operator
bool Mystring::operator!= (const Mystring& S) const {
  if (this->isNotEqual(S)==1) {
    return true;
  } else {
    return false;
```

```
  }
}

// Overloading > operator
bool Mystring::operator> (const Mystring& S) const {
  if (this->isGreaterThan(S)==1) {
    return true;
  } else {
    return false;
  }
}

// Overloading < operator
bool Mystring::operator< (const Mystring& S) const {
  if (this->isLessThan(S)==1) {
    return true;
  } else {
    return false;
  }
}

// Overloading == operator
bool Mystring::operator== (const Mystring& S) const {
  if (this->isEqual(S)==1) {
    return true;
  } else {
    return false;
  }
}

// Overloading [] operator
char& Mystring::operator[](int i) const {
  if (i >= this->length() || i < 0) {
    throw std::out_of_range("\nERROR: get_char: the position is out of
boundary.");
  } else {
    return (charsM[i]);
  }
}
```

Exercise B:

Code:

main.cpp

```cpp
#include "graphicsWorld.cpp"

int main(){
    GraphicsWorld world;
    world.run();

    return 0;
}
```

point.h and point.cpp

```cpp
#ifndef POINT_H
#define POINT_H

class Point {
public:
    // Constructor
    Point(double x, double y);

    // Display function
    void display() const;

    // Getter functions
    double getX() const;
    double getY() const;
    void setX(double x);
    void setY(double y);
    int getId() const;

    // Static function to calculate distance between two points
    static double distance(const Point& p1, const Point& p2);

    // Member function to calculate distance between this point and another
point
    double distance(const Point& other) const;
```

```
    // Function to get the total count of Point objects
    static int counter();

private:
    double x;
    double y;
    int id;
    static int num;
};

#endif // POINT_H
```

```
#include "point.h"
#include <iostream>
#include <cmath>

using namespace std;

// Global ID Starting Point
int Point::num = 1000;

// Constructor
Point::Point(double xi, double yi){
    x = xi;
    y = yi;
    id = num++;
}

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

// Getters
double Point::getX() const {
    return x;
```

```cpp
}

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

int Point::getId() const {
    return id;
}

// Setters
void Point::setX(double x){
    this->x = x;
}

void Point::setY(double y){
    this->y = y;
}

// Distance between 2 points
double Point::distance(const Point& p1, const Point& p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return sqrt(dx * dx + dy * dy);
}

// Distance between this Point and another
double Point::distance(const Point& other) const {
    double dx = x - other.x;
    double dy = y - other.y;
    return sqrt(dx * dx + dy * dy);
}

// Shape Count = Global IDs - 1000
int Point::counter() {
    return num - 1000;
}
```

shape.h and shape.cpp

```
#ifndef SHAPE_H
#define SHAPE_H

#include "point.h"
using namespace std;

class Shape {
public:
    // Constructor
    Shape(const Point& origin, const char* shapeName);

    // Copy constructor
    Shape(const Shape& other);

    // Assignment operator
    Shape& operator=(const Shape& other);

    // Destructor
    ~Shape();

    // Getter for origin (reference to a const Point)
    const Point& getOrigin() const;

    // Getter for shapeName
    const char* getName() const;

    // Display function
    void display() const;

    // Distance function between two shapes
    double distance(Shape& other);

    // Static distance function between two shapes
    static double distance(Shape& shape1, Shape& shape2);

    // Move function
    void move(double dx, double dy);
```

```
private:
    Point origin;
    char* shapeName;
};

#endif // SHAPE_H
```

```
#include "shape.h"
#include <iostream>
#include <cstring>
#include <cmath>

using namespace std;

// Constructor
Shape::Shape(const Point& origin, const char* shapeName) : origin(origin) {
    this->shapeName = new char[strlen(shapeName) + 1];
    strcpy(this->shapeName, shapeName);
}

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

// Assignment operator
Shape& Shape::operator=(const Shape& other) {
    if (this != &other) {
        delete[] shapeName;

        origin = other.origin;
```

```
        if (other.shapeName) {
            shapeName = new char[strlen(other.shapeName) + 1];
            strcpy(shapeName, other.shapeName);
        } else {
            shapeName = nullptr;
        }
    }
    return *this;
}

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

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

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

// Display function
void Shape::display() const {
    cout << "Shape Name: " << shapeName << endl;
    cout << "X-coordinate: " << origin.getX() << endl;
    cout << "Y-coordinate: " << origin.getY() << endl;
}

// Distance function between this shape and another
double Shape::distance(Shape& other) {
    return origin.distance(other.getOrigin());
}

// Distance function between two shapes
double Shape::distance(Shape& shape1, Shape& shape2) {
```

```
    return shape1.origin.distance(shape2.origin);
}

// Move function
void Shape::move(double dx, double dy) {
    origin.setX(origin.getX() + dx);
    origin.setY(origin.getY() + dy);
}
```

square.h and square.cpp

```
#ifndef SQUARE_H
#define SQUARE_H

#include "shape.h"

class Square : public Shape {
public:
    // Constructor
    Square(const Point& origin, double side_a, const char* shapeName);

    // Overloaded Constructor
    Square(double x, double y, double side_a, const char* shapeName);

    // Copy constructor
    Square(const Square& other);

    // Assignment operator
    Square& operator=(const Square& other);

    // Destructor
    ~Square();

    // Getter and Setter for side_a
    double getSideA() const;
    void setSideA(double side_a);

    // Area function
```

```
    virtual double area() const;

    // Perimeter function
    virtual double perimeter() const;

    // Display function
    virtual void display() const;

protected:
    double side_a;
};

#endif // SQUARE_H
```

```
#include "square.h"
#include <iostream>

// Constructor
Square::Square(const Point& origin, double side_a, const char* shapeName):
Shape(origin, shapeName), side_a(side_a) {
}

Square::Square(double x, double y, double side_a, const char* shapeName):
Shape(Point(x, y), shapeName), side_a(side_a) {
}

// Copy constructor
Square::Square(const Square& other) : Shape(other) {
    // Copy any dynamic resources from 'other' to this object (if any)
    side_a = other.side_a;
}

// Assignment operator
Square& Square::operator=(const Square& other) {
    if (this != &other) {
        Shape::operator=(other);
        side_a = other.side_a;
    }
```

```
    return *this;
}

// Destructor
Square::~Square() {
    // Nothing to Release
}

// Getters
double Square::getSideA() const {
    return side_a;
}

// Setters
void Square::setSideA(double side_a) {
    this->side_a = side_a;
}

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

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

// Display function
void Square::display() const {
    std::cout << "Square Name: " << getName() << std::endl;
    std::cout << "X-coordinate: " << getOrigin().getX() << std::endl;
    std::cout << "Y-coordinate: " << getOrigin().getY() << std::endl;
    std::cout << "Side a: " << side_a << std::endl;
    std::cout << "Area: " << area() << std::endl;
    std::cout << "Perimeter: " << perimeter() << std::endl;
}
```

rectangle.h and rectangle.cpp

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "Square.h"

class Rectangle : public Square {
public:
    // Constructors
    Rectangle(const Point& origin, double side_a, double side_b, const char*
shapeName);

    Rectangle(double x, double y, double side_a, double side_b, const char*
shapeName);

    // Copy constructor
    Rectangle(const Rectangle& other);

    // Assignment operator
    Rectangle& operator=(const Rectangle& other);

    // Destructor
    ~Rectangle();

    // Getter and Setter for side_b
    double getSideB() const;
    void setSideB(double side_b);

    // Area function
    double area() const;

    // Perimeter function
    double perimeter() const;

    // Display function
    void display() const;

private:
```

```
    double side_b;
};

#endif // RECTANGLE_H
```

```
#include "Rectangle.h"
#include <iostream>

// Constructor
Rectangle::Rectangle(const Point& origin, double side_a, double side_b, const
char* shapeName)
    : Square(origin, side_a, shapeName), side_b(side_b) {
}

Rectangle::Rectangle(double x, double y, double side_a, double side_b, const
char* shapeName)
    : Square(Point(x,y), side_a, shapeName), side_b(side_b) {
}

//Copy Constructor
Rectangle::Rectangle(const Rectangle& other) : Square(other) {
    side_b = other.side_b;
}

// Assignment operator
Rectangle& Rectangle::operator=(const Rectangle& other) {
    if (this != &other) {
        Square::operator=(other);

        side_b = other.side_b;
    }
    return *this;
}

// Destructor
Rectangle::~Rectangle() {
    // No dynamic memory to release
}
```

```cpp
// Getters
double Rectangle::getSideB() const {
    return side_b;
}

// Setters
void Rectangle::setSideB(double side_b) {
    this->side_b = side_b;
}

// Area function
double Rectangle::area() const {
    return side_a * side_b;
}

// Perimeter function
double Rectangle::perimeter() const {
    return 2 * (side_a + side_b);
}

// Display function
void Rectangle::display() const {
    std::cout << "Rectangle Name: " << getName() << std::endl;
    std::cout << "X-coordinate: " << getOrigin().getX() << std::endl;
    std::cout << "Y-coordinate: " << getOrigin().getY() << std::endl;
    std::cout << "Side a: " << side_a << std::endl;
    std::cout << "Side b: " << side_b << std::endl;
    std::cout << "Area: " << area() << std::endl;
    std::cout << "Perimeter: " << perimeter() << std::endl;
}
```

GraphicsWorld.h and GraphicsWorld.cpp

```cpp
#ifndef GRAPHICSWORLD_H
#define GRAPHICSWORLD_H

#include "Point.cpp"
```

```
#include "Shape.cpp"
#include "Square.cpp"
#include "Rectangle.cpp"

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

#endif // GRAPHICSWORLD_H
```

```
#include "graphicsWorld.h"
#include <iostream>

void GraphicsWorld::run() {
#if 1 // Change 0 to 1 to test Point
    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

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

#if 1 // Change 0 to 1 to test Rectangle
    std::cout << "\nTesting Functions in class Rectangle:";

    Rectangle a(5, 7, 12, 15, "RECTANGLE A");
    a.display();
```

```
    Rectangle b(16, 7, 8, 9, "RECTANGLE B");
    b.display();

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

    Rectangle rec1 = a;
    rec1.display();

    std::cout << "\nTesting assignment operator in class Rectangle:" <<
std::endl;
    Rectangle rec2(3, 4, 11, 7, "RECTANGLE rec2");
    rec2.display();
    rec2 = a;
    a.setSideB(200);
    a.setSideA(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.setSideB(300);
```

```
    a.setSideA(400);

    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: 100\n"
              << "Side b: 200\n"
              << "Area: 20000\n"
              << "Perimeter: 600\n";

    std::cout << "\nIf it doesn't, there is a problem with your assignment
operator.\n"
              << std::endl;

    rec3.display();
#endif // end of block to test Rectangle

#if 1 // Change 0 to 1 to test using an array of pointers and polymorphism
    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;

    for (int i = 0; i < 4; ++i) {
        sh[i]->display();
    }
#endif // end of block to test an array of pointers and polymorphism
}
```

Output:

```
C:\Users\dominic\Desktop\Schoolwork\2023_Q4_Fall\ENSF480\ENSF-480-Labs\Lab2\exB>.\main.exe

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
Side a: 12
Side b: 15
Area: 180
Perimeter: 54
Rectangle Name: RECTANGLE B
X-coordinate: 16
Y-coordinate: 7
Side a: 8
Side b: 9
Area: 72
Perimeter: 34

Distance between square a and b is: 11
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 12
Side b: 15
Area: 180
Perimeter: 54

Testing assignment operator in class Rectangle:
Rectangle Name: RECTANGLE rec2
X-coordinate: 3
Y-coordinate: 4
Side a: 11
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
Side a: 12
Side b: 15
Area: 180
Perimeter: 54

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

Expected to display the following values for object rec2:
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 assignment operator.

Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 100
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
```