Lab 03 Report **Completed by: Dominic Choi 30109955**
Nathan Ante 30157706

Exercise A:

Output of main.cpp:

```
---------------------- Start of Lab 3 ----------------------

Testing Functions in class Circle:
Circle Name: CIRCLE C
X-coordinate: 3
Y-coordinate: 5
Radius: 9
Area: 254.469
Circumference: 56.5487
the area of CIRCLE C is: 254.469
the perimeter of CIRCLE C is: 56.5487

The distance between rectangle a and circle c is: 2.82843
CurveCut Name: CurveCut rc
X-coordinate: 6
Y-coordinate: 5
Width: 10
Length: 12
Radius of the cut: 9
the area of CurveCut rc is: 56.3827
the perimeter of CurveCut rc is: 26

The distance between rc and c is: 3
Square Name: SQUARE - S
X-coordinate: 5
Y-coordinate: 7
Side a: 12
Area: 144
Perimeter: 48

the area of SQUARE - S is: 144

the perimeter of SQUARE - S is: 48
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 400
Side b: 300
Area: 120000
Perimeter: 1400

the area of RECTANGLE A is: 120000
the perimeter of SQUARE - S is: 1400
Circle Name: CIRCLE C
X-coordinate: 3
Y-coordinate: 5
Radius: 9
Area: 254.469
Circumference: 56.5487

the area of CIRCLE C is: 254.469
the circumference of CIRCLE C is: 56.5487
CurveCut Name: CurveCut rc
X-coordinate: 6
Y-coordinate: 5
Width: 10
Length: 12
Radius of the cut: 9

the area of CurveCut rc is: 56.3827
the perimeter of CurveCut rc is: 26

Testing copy constructor in class CurveCut:
CurveCut Name: CurveCut rc
X-coordinate: 6
Y-coordinate: 5
Width: 10
Length: 12
Radius of the cut: 9

Testing assignment operator in class CurveCut:
CurveCut Name: CurveCut cc2
X-coordinate: 2
Y-coordinate: 5
Width: 100
Length: 12
Radius of the cut: 9
CurveCut Name: CurveCut rc
X-coordinate: 6
Y-coordinate: 5
Width: 10
Length: 12
Radius of the cut: 9
```

main.cpp

```cpp
#include "GraphicsWorld.cpp"

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

    return 0;
}
```

Point.h
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
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
    virtual void display() const;

    virtual double area() const = 0;
    virtual double perimeter() const = 0;

    // 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);
}
```

Circle.h
Circle.cpp

```
#ifndef CIRCLE_H
#define CIRCLE_H

#define M_PI 3.14159265358979323846

#include "Shape.h"

class Circle : public virtual Shape {
private:
    double radius;

public:
    // Constructors
    Circle(const Point& origin, double radius, const char* shapeName);
    Circle(double x, double y, double radius, const char* shapeName);

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

    // Assignment operator
    Circle& operator=(const Circle& other);
```

```
    // Getters
    double getRadius() const;

    // Setters
    void setRadius(double radius);

    // Area function
    double area() const;

    // Perimeter function (circumference)
    double perimeter() const;

    // Display function
    void display() const;

    // Destructor
    ~Circle();
};

#endif // CIRCLE_H
```

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

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

Circle::Circle(double x, double y, double radius, const char* shapeName)
    : Shape(Point(x, y), shapeName), radius(radius) {
}

// Copy constructor
Circle::Circle(const Circle& other) : Shape(other) {
    radius = other.radius;
```

```cpp
}

// Assignment operator
Circle& Circle::operator=(const Circle& other) {
    if (this != &other) {
        Shape::operator=(other);
        radius = other.radius;
    }
    return *this;
}

// Getters
double Circle::getRadius() const {
    return radius;
}

// Setters
void Circle::setRadius(double radius) {
    this->radius = radius;
}

// Area function
double Circle::area() const {
    return M_PI * radius * radius;
}

// Perimeter function (circumference)
double Circle::perimeter() const {
    return 2 * M_PI * radius;
}

// Display function
void Circle::display() const {
    std::cout << "Circle Name: " << getName() << std::endl;
    std::cout << "X-coordinate: " << getOrigin().getX() << std::endl;
    std::cout << "Y-coordinate: " << getOrigin().getY() << std::endl;
    std::cout << "Radius: " << radius << std::endl;
    std::cout << "Area: " << area() << std::endl;
```

```
    std::cout << "Circumference: " << perimeter() << std::endl;
}

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

Square.h
Square.cpp

```
#ifndef SQUARE_H
#define SQUARE_H

#include "Shape.h"

class Square : public virtual 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
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), Shape(origin,
shapeName) {
}

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), Shape(Point(x,y),
shapeName) {
}

//Copy Constructor
Rectangle::Rectangle(const Rectangle& other) : Square(other), Shape(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
```

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

// 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;
}
```

CurveCut.h
CurveCut.cpp

```
#ifndef CURVECUT_H
#define CURVECUT_H

#include "Rectangle.h"
#include "Circle.h"

class CurveCut : public Rectangle, public Circle{
public:
    // Constructor
    CurveCut(const Point& origin, double side_a, double side_b,
             double radius, const char* shapeName);

    CurveCut(double x, double y, double width, double length, double radius,
const char* shapeName);

    CurveCut& operator=(const CurveCut& other);
    // Area function
    double area() const;

    const char* getName();

    // Perimeter function
    double perimeter() const;

    double distance(Shape &other) const;

    // Display function
    void display() const;

    // Destructor
    ~CurveCut();
};

#endif // CURVECUT_H
```

```cpp
#include "CurveCut.h"
#include <iostream>
#include <cmath>
#include <cstring>

using namespace std;

// Constructor
CurveCut::CurveCut(const Point& origin, double width, double length, double
radius, const char* shapeName)
    : Rectangle(origin, width, length, shapeName), Circle(origin, radius,
shapeName), Shape(origin, shapeName){
    // Check if the radius is valid
    if (radius > min(width, length)) {
        cerr << "Error: The radius must be less than or equal to the smaller of
width and length." << endl;
        exit(1); // Terminate the program
    }
}

CurveCut::CurveCut(double x, double y, double width, double length, double
radius, const char* shapeName)
    : CurveCut(Point(x, y), width, length, radius, shapeName){
    // Check if the radius is valid
    if (radius > min(width, length)) {
        cerr << "Error: The radius must be less than or equal to the smaller of
width and length." << endl;
        exit(1); // Terminate the program
    }
}

CurveCut& CurveCut::operator=(const CurveCut& other) {
    if (this != &other) {
        Rectangle::operator=(other);
        Circle::operator=(other);
    }
    return *this;
}
```

```cpp
// Area function
double CurveCut::area() const {
    double rectArea = this->getSideA() * this->getSideB();
    double circleArea = M_PI * this->getRadius() * this->getRadius();
    return rectArea - circleArea/4;
}

const char* CurveCut::getName(){
    return Rectangle::getName();
}

// Perimeter function
double CurveCut::perimeter() const {
    double rectPerimeter = 2 * (this->getSideA() + this->getSideB());
    double circlePerimeter = 1/2 * M_PI * this->getRadius();
    return (rectPerimeter - 2*this->getRadius()) + circlePerimeter/4;
}

double CurveCut::distance(Shape& other) const {
    return Rectangle::getOrigin().distance(other.getOrigin());
}

// Display function
void CurveCut::display() const {
    cout << "CurveCut Name: " << Rectangle::getName() << endl;
    cout << "X-coordinate: " << Rectangle::getOrigin().getX() << endl;
    cout << "Y-coordinate: " << Rectangle::getOrigin().getY() << endl;
    cout << "Width: " << getSideA() << endl;
    cout << "Length: " << getSideB() << endl;
    cout << "Radius of the cut: " << getRadius() << endl;
}

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

GraphicsWorld.h
GraphicsWorld.cpp

```h
#ifndef GRAPHICSWORLD_H
#define GRAPHICSWORLD_H

#include "Point.cpp"
#include "Shape.cpp"
#include "Square.cpp"
#include "Rectangle.cpp"
#include "Circle.cpp"
#include "CurveCut.cpp"

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

#endif // GRAPHICSWORLD_H
```

```cpp
#include "GraphicsWorld.h"
#include <iostream>

void GraphicsWorld::run() {
#if 0 // Change 0 to 1 to test Point
    Point m(6, 8);
    Point n(6, 8);
    n.setX(9);
    cout << "\nExpected to display the distance between m and n is: 3";
    cout << "\nThe distance between m and n is: " << m.distance(n);
    cout << "\nExpected second version of the distance function also print: 3";
    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
    cout << "\n\nTesting Functions in class Square:" << 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
    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);
    cout << "\nDistance between square a and b is: " << d << endl;

    Rectangle rec1 = a;
    rec1.display();

    cout << "\nTesting assignment operator in class Rectangle:" << endl;
    Rectangle rec2(3, 4, 11, 7, "RECTANGLE rec2");
    rec2.display();
    rec2 = a;
    a.setSideB(200);
    a.setSideA(100);

    cout << "\nExpected to display the following values for object rec2: " <<
endl;
    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";

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

```
    rec2.display();

    cout << "\nTesting copy constructor in class Rectangle:" << endl;
    Rectangle rec3(a);
    rec3.display();
    a.setSideB(300);
    a.setSideA(400);

    cout << "\nExpected to display the following values for object rec2: " <<
endl;
    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";

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

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

#if 0 // Change 0 to 1 to test using an array of pointers and polymorphism
    cout << "\nTesting array of pointers and polymorphism:" << 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
```

```
#if 1
    cout << "\n----------------------- Start of Lab 3
-----------------------" << endl;
    cout << "\nTesting Functions in class Circle:" << endl;
    Circle c (3, 5, 9, "CIRCLE C");
    c.display();
    cout << "the area of " << c.getName() <<" is: "<< c.area() << endl;
    cout << "the perimeter of " << c.getName() << " is: "<< c.perimeter() <<
endl;
    d = a.Rectangle::distance(c);
    cout << "\nThe distance between rectangle a and circle c is: " << d <<
endl;

    CurveCut rc (6, 5, 10, 12, 9, "CurveCut rc");
    rc.display();
    cout << "the area of " << rc.getName() <<" is: "<< rc.area() << endl;
    cout << "the perimeter of " << rc.getName() << " is: "<< rc.perimeter() <<
endl;
    d = rc.distance(c);
    cout << "\nThe distance between rc and c is: " << d << endl;

    // Using array of Shape pointers:
    Shape* sh[4];
    sh[0] = &s;
    sh[1] = &a;
    sh [2] = &c;
    sh [3] = &rc;
    sh [0]->display();
    cout << "\nthe area of "<< sh[0]->getName() << " is: "<< sh[0] ->area() <<
endl;
    cout << "\nthe perimeter of " << sh[0]->getName () << " is: "<<
sh[0]->perimeter() << endl;
    sh [1]->display();
    cout << "\nthe area of "<< sh[1]->getName() << " is: "<< sh[1] ->area();
    cout << "\nthe perimeter of " << sh[0]->getName () << " is: "<<
sh[1]->perimeter() << endl;
    sh [2]->display();
    cout << "\nthe area of "<< sh[2]->getName() << " is: "<< sh[2] ->area();
```

```
    cout << "\nthe circumference of " << sh[2]->getName ()<< " is: "<<
sh[2]->perimeter() << endl;
    sh [3]->display();
    cout << "\nthe area of "<< sh[3]->getName() << " is: "<< sh[3]->area();
    cout << "\nthe perimeter of " << sh[3]->getName () << " is: "<<
sh[3]->perimeter() << endl;

    cout << "\nTesting copy constructor in class CurveCut:" << endl;
    CurveCut cc = rc;
    cc.display();

    cout << "\nTesting assignment operator in class CurveCut:" << endl;
    CurveCut cc2(2, 5, 100,  12, 9,  "CurveCut cc2");
    cc2.display();
    cc2 = cc;
    cc2.display();
#endif
}         // END OF FUNCTION run
```

Exercise B:
Output

```
The first element of vector x contains: 999
Testing an <int> Vector:


Testing sort
-77
88
999

Testing Prefix --:
999
88
-77

Testing Prefix ++:
88
999
-77

Testing Postfix --
-77
999
88
Testing a <Mystring> Vector:


Testing sort
All
Bar
Foo

Testing Prefix --:
Foo
Bar
All

Testing Prefix ++:
Bar
Foo
All

Testing Postfix --
All
Foo
Bar
Testing a <char *> Vector:


Testing sort
Orange
Pear
Apple
Prgram Terminated Successfully.
```

iterator.cpp

```cpp
// iterator.cpp
// ENSF 480 - Fall 2022 - Lab 3, Ex B

#include <iostream>
#include <assert.h>
#include "mystring2.h"
#include <algorithm>

using namespace std;

template <class T>
class Vector {
public:

  class VectIter{
    friend class Vector;
  private:
    Vector *v; // points to a vector object of type T
    int index;    // represents the subscript number of the vector's
                  // array.
  public:
    VectIter(Vector& x);

    T operator++();
    //PROMISES: increments the iterator's index and return the
    //          value of the element at the index position. If
    //          index exceeds the size of the array it will
    //          be set to zero. Which means it will be circulated
    //          back to the first element of the vector.

    T operator++(int);
    // PROMISES: returns the value of the element at the index
    //           position, then increments the index. If
    //           index exceeds the size of the array it will
    //           be set to zero. Which means it will be circulated
    //           back to the first element of the vector.
```

```
    T operator--();
    // PROMISES: decrements the iterator index, and return the
    //           the value of the element at the index. If
    //           index is less than zero it will be set to the
    //           last element in the aray. Which means it will be
    //           circulated to the last element of the vector.

    T operator--(int);
    // PRIMISES: returns the value of the element at the index
    //           position, then decrements the index. If
    //           index is less than zero it will be set to the
    //           last element in the aray. Which means it will be
    //           circulated to the last element of the vector.

    T operator *();
    // PRIMISES: returns the value of the element at the current
    //           index position.
  };

  Vector(int sz);
  ~Vector();

  T & operator[](int i);
  // PRIMISES: returns existing value in the ith element of
  //           array or sets a new value to  the ith element in
  //           array.

  void ascending_sort();
  // PRIMISES: sorts the vector values in ascending order.

private:
  T *array;               // points to the first element of an array of T
  int size;               // size of array
  void swap(T&, T&); // swaps the values of two elements in array
public:
};

template <class T>
```

```
void Vector<T>::ascending_sort()
{
  for(int i=0; i< size-1; i++)
    for(int j=i+1; j < size; j++)
      if(array[i] > array[j])
        swap(array[i], array[j]);
}

template <class T>
void Vector<T>::swap(T& a, T& b)
{
  T tmp = a;
  a = b;
  b = tmp;
}

template <class T>
T Vector<T>::VectIter::operator *()
{
  return v -> array[index];
}

template <class T>
Vector<T>::VectIter::VectIter(Vector& x)
{
  v = &x;
  index = 0;
}


template <class T>
Vector<T>::Vector(int sz)
{
  size=sz;
  array = new T [sz];
  assert (array != NULL);
}
```

```cpp
template <class T>
Vector<T>::~Vector()
{
  delete [] array;
  array = NULL;
}


template <class T>
T & Vector<T>::operator [] (int i)
{
  return array[i];
}

/*-----------------------------Function Definitions start
here-----------------------------*/
template <class T>
T Vector<T>::VectIter::operator++() {
  if (++index > v->size - 1) {
    index = 0;
  }
  return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator++(int a) {
  T returnVal = v->array[index + a];

  if (++index > v->size - 1) {
    index = 0;
  }

  return returnVal;
}

template <class T>
T Vector<T>::VectIter::operator--() {
  if (--index < 0) {
```

```
    index = v->size - 1;
  }
  return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator--(int a) {
  T returnVal = v->array[index + a];
  if (--index < 0) {
    index = v->size - 1;
  }
  return returnVal;
}

/*------------------------------Function Definitions end
here------------------------------*/

int main()
{

 Vector<int> x(3);
 x[0] = 999;
 x[1] = -77;
 x[2] = 88;

 Vector<int>::VectIter iter(x);

 cout << "\n\nThe first element of vector x contains: " << *iter;

 // the code between the  #if 0 and #endif is ignored by
 // compiler. If you change it to #if 1, it will be compiled

#if 1
  cout << "\nTesting an <int> Vector: " << endl;

  cout << "\n\nTesting sort";
  x.ascending_sort();
```

```
for (int i=0; i<3 ; i++)
  cout << endl << iter++;

cout << "\n\nTesting Prefix --:";
for (int i=0; i<3 ; i++)
  cout << endl << --iter;

cout << "\n\nTesting Prefix ++:";
for (int i=0; i<3 ; i++)
  cout << endl << ++iter;

cout << "\n\nTesting Postfix --";
for (int i=0; i<3 ; i++)
  cout << endl << iter--;

cout << endl;

cout << "Testing a <Mystring> Vector: " << endl;
Vector<Mystring> y(3);
y[0] = "Bar";
y[1] = "Foo";
y[2] = "All";;

Vector<Mystring>::VectIter iters(y);

cout << "\n\nTesting sort";
y.ascending_sort();

for (int i=0; i<3 ; i++)
  cout << endl << iters++;

cout << "\n\nTesting Prefix --:";
for (int i=0; i<3 ; i++)
  cout << endl << --iters;

cout << "\n\nTesting Prefix ++:";
for (int i=0; i<3 ; i++)
  cout << endl << ++iters;
```

```
  cout << "\n\nTesting Postfix --";
  for (int i=0; i<3 ; i++)
    cout << endl << iters--;

  cout << endl; cout << "Testing a <char *> Vector: " << endl;
  Vector<const char*> z(3);
  z[0] = "Orange";
  z[1] = "Pear";
  z[2] = "Apple";

  Vector<const char*>::VectIter iterchar(z);

  cout << "\n\nTesting sort";
  z.ascending_sort();

  for (int i=0; i<3 ; i++)
    cout << endl << iterchar++;

#endif
  cout << "\nPrgram Terminated Successfully." << endl;

  return 0;
}
```

mystring2.cpp

```
//  mystring2.cpp
// ENSF 480 - Fall 2022 - Lab 3, Ex B and C

#include "mystring2.h"
#include <string.h>
#include <iostream>
using namespace std;

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

```
  charsM[0] = '\0';
  lengthM = 0;
}

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

Mystring::Mystring(int n)
  : lengthM(0), charsM(new char[n])
{
  charsM[0] = '\0';
}

Mystring::Mystring(const Mystring& source):
  lengthM(source.lengthM), charsM(new char[source.lengthM+1])
{
  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];
  strcpy(charsM,S.charsM);
  return *this;
}
Mystring& Mystring::append(const Mystring& other)
{
```

```
  char *tmp = new char [lengthM + other.lengthM + 1];
  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];

    strcpy(charsM, s);
}

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


int Mystring::operator> (const Mystring& S) const {
  return (strcmp(charsM, S.charsM) > 0);
}
```

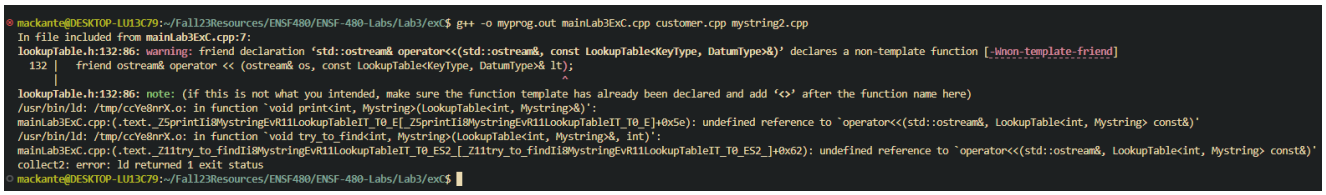
mystring2.h

```
//File: mystring2.h
// ENSF 480 - Fall 2022 - Lab 3, Ex B and C


#ifndef MYSTRING_H
#define MYSTRING_H
```

```
#include <ostream>

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.

  //
  friend std::ostream& operator << (std::ostream& os, const Mystring& rhs);
  int operator > (const Mystring& rhs) 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
```

Exercise C:
Output - doesn't work help



lookupTable.h

```
// LookupTable.h
// ENSF 480 - Lab 3, Ex C

#ifndef LOOKUPTABLE_H
#define LOOKUPTABLE_H
#include <iostream>
using namespace std;

// class LookupTable: 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 LookupTable has an embedded iterator class that allows users
//    of the class to traverse trhough the list and  have acess to each
//    node.

#include "customer.h"

//    In this version of the LookupTable a new struct type called Pair
//    is introduced which represents a key/data pair.

typedef int KeyType;
typedef Customer DatumType;
template<class KeyType, class DatumType> class LookupTable;

template<class KeyType, class DatumType>
struct Pair
```

```
{
  Pair(KeyType keyA, DatumType datumA):key(keyA), datum(datumA)
  {
  }

  KeyType key;
  DatumType datum;
};

template<class KeyType, class DatumType>
class LT_Node {
  friend class LookupTable<KeyType, DatumType>;
private:
  Pair<KeyType, DatumType> pairM;
  LT_Node<KeyType, DatumType> *nextM;

  // This ctor should be convenient in insert and copy operations.
  LT_Node<KeyType, DatumType>(const Pair<KeyType, DatumType>& pairA,
LT_Node<KeyType, DatumType> *nextA);
};

template<class KeyType, class DatumType>
class LookupTable {
 public:

  // Nested class
  class Iterator {
    friend class LookupTable;
    LookupTable *LT;
// LT_Node<KeyType, DatumType>* cursor;

  public:
    Iterator():LT(0){}
    Iterator(LookupTable<KeyType, DatumType>& x): LT(&x){}
    const DatumType&  operator *();
    const DatumType& operator ++();
    const DatumType& operator ++(int);
    int operator !();
```

```
  void step_fwd(){  assert(LT->cursor_ok());
  LT->step_fwd();}
};

LookupTable();
LookupTable(const LookupTable<KeyType, DatumType>& source);
LookupTable& operator =(const LookupTable<KeyType, DatumType>& rhs);
~LookupTable();

LookupTable& begin();

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

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

void insert(const Pair<KeyType, DatumType>& pariA);
// 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 KeyType& 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 KeyType& 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.

  friend ostream& operator << (ostream& os, const LookupTable<KeyType,
DatumType>& lt);

 private:
  int sizeM;
  LT_Node<KeyType, DatumType> *headM;
  LT_Node<KeyType, DatumType> *cursorM;

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

  void copy(const LookupTable<KeyType, DatumType>& 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

/*-----------------------------------------------------------------------------
-----------------------------------------*/
template<class KeyType, class DatumType>
LookupTable<KeyType, DatumType>& LookupTable<KeyType, DatumType>::begin(){
  cursorM = headM;
  return *this;
}

template<class KeyType, class DatumType>
LT_Node<KeyType, DatumType>::LT_Node(const Pair<KeyType, DatumType>& pairA,
LT_Node<KeyType, DatumType> *nextA)
  : pairM(pairA), nextM(nextA)
{
}

template<class KeyType, class DatumType>
LookupTable<KeyType, DatumType>::LookupTable()
  : sizeM(0), headM(0), cursorM(0)
{
}

template<class KeyType, class DatumType>
LookupTable<KeyType, DatumType>::LookupTable(const LookupTable<KeyType,
DatumType>& source)
{
  copy(source);
}

template<class KeyType, class DatumType>
LookupTable<KeyType, DatumType>& LookupTable<KeyType, DatumType>::operator
=(const LookupTable<KeyType, DatumType>& rhs)
{
```

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

template<class KeyType, class DatumType>
LookupTable<KeyType, DatumType>::~LookupTable()
{
  destroy();
}

template<class KeyType, class DatumType>
int LookupTable<KeyType, DatumType>::size() const
{
  return sizeM;
}

template<class KeyType, class DatumType>
int LookupTable<KeyType, DatumType>::cursor_ok() const
{
  return cursorM != 0;
}

template<class KeyType, class DatumType>
const KeyType& LookupTable<KeyType, DatumType>::cursor_key() const
{
  assert(cursor_ok());
  return cursorM->pairM.key;
}

template<class KeyType, class DatumType>
const DatumType& LookupTable<KeyType, DatumType>::cursor_datum() const
{
  assert(cursor_ok());
  return cursorM->pairM.datum;
}
```

```
template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::insert(const Pair<KeyType, DatumType>&
pairA)
{
  // Add new node at head?
  if (headM == 0 || pairA.key < headM->pairM.key) {
    headM = new LT_Node<KeyType, DatumType>(pairA, headM);
    sizeM++;
  }

  // Overwrite datum at head?
  else if (pairA.key == headM->pairM.key)
    headM->pairM.datum = pairA.datum;

  // Have to search ...

  else {
    LT_Node<KeyType, DatumType>* before= headM;
    LT_Node<KeyType, DatumType>* after=headM->nextM;

    while(after!=NULL && (pairA.key > after->pairM.key))
      {
  before=after;
  after=after->nextM;
      }

    if(after!=NULL && pairA.key == after->pairM.key)
      {
  after->pairM.datum = pairA.datum;
      }
    else
      {
  before->nextM = new LT_Node<KeyType, DatumType> (pairA, before->nextM);
  sizeM++;
      }
  }
}
```

```
template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::remove(const KeyType& keyA)
{

  if (headM == 0 || keyA < headM->pairM.key)
    return;

  LT_Node<KeyType, DatumType>* doomed_node = 0;
  if (keyA == headM->pairM.key) {
    doomed_node = headM;
    headM = headM->nextM;
    sizeM--;
  }
  else {
    LT_Node<KeyType, DatumType> *before = headM;
    LT_Node<KeyType, DatumType> *maybe_doomed = headM->nextM;
    while(maybe_doomed != 0 && keyA > maybe_doomed->pairM.key) {
      before = maybe_doomed;
      maybe_doomed = maybe_doomed->nextM;
    }

    if (maybe_doomed != 0 && maybe_doomed->pairM.key == keyA) {
      doomed_node = maybe_doomed;
      before->nextM = maybe_doomed->nextM;
      sizeM--;
    }
  }
  delete doomed_node;           // Does nothing if doomed_node == 0.
}

template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::find(const KeyType& keyA)
{
  LT_Node<KeyType, DatumType> *ptr=headM;
  while (ptr != NULL && ptr->pairM.key != keyA)
    {
     ptr=ptr->nextM;
```

```
    }

   cursorM = ptr;
}

template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::go_to_first()
{
  cursorM = headM;
}

template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::step_fwd()
{
  assert(cursor_ok());
  cursorM = cursorM->nextM;
}

template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::make_empty()
{
  destroy();
  sizeM = 0;
  cursorM = 0;
}

template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::destroy()
{

  LT_Node<KeyType, DatumType> *ptr = headM;
  while (ptr!=NULL)
    {
      headM=headM->nextM;
      delete ptr;
      ptr=headM;

    }
```

```
  cursorM = NULL;
  sizeM=0;
}

template<class KeyType, class DatumType>
void LookupTable<KeyType, DatumType>::copy(const LookupTable<KeyType,
DatumType>& source)
{

  headM=0;
  cursorM =0;

  if(source.headM ==0)
    return;

  for(LT_Node<KeyType, DatumType> *p = source.headM; p != 0; p=p->nextM)
    {
      insert(Pair<KeyType, DatumType> (p->pairM.key, p->pairM.datum));
      if(source.cursorM == p)
  find(p->pairM.key);
    }

}

template<class KeyType, class DatumType>
ostream& operator << (ostream& os, const LookupTable<KeyType, DatumType>& lt)
{
  if (lt.cursor_ok())
    os << lt.cursor_key() << "  " << lt.cursor_datum();
  else
    os << "Not Found.";

  return os;
}

template<class KeyType, class DatumType>
const DatumType& LookupTable<KeyType, DatumType>::Iterator::operator *()
{
```

```
  assert(LT ->cursor_ok());
  return LT->cursor_datum();
}

template<class KeyType, class DatumType>
const DatumType& LookupTable<KeyType, DatumType>::Iterator::operator ++()
{
  assert(LT->cursor_ok());
  const DatumType & x = LT->cursor_datum();
  LT->step_fwd();
  return x;
}

template<class KeyType, class DatumType>
const DatumType& LookupTable<KeyType, DatumType>::Iterator::operator ++(int)
{
  assert(LT->cursor_ok());

  LT->step_fwd();
  return LT->cursor_datum();
}

template<class KeyType, class DatumType>
int LookupTable<KeyType, DatumType>::Iterator::operator!()
{
  return (LT->cursor_ok());
}
```

mainLab3ExC.cpp

```
// ENSF 480 - Lab 3, Ex C
// M. Moussavi

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

```cpp
#include "customer.h"
#include <cstring>
using namespace std;

template<class KeyType, class DatumType>
void print(LookupTable<KeyType, DatumType>& lt);

template<class KeyType, class DatumType>
void try_to_find(LookupTable<KeyType, DatumType>& lt, KeyType key);

void test_Customer();

//Uncomment the following function calls when ready to test template class
LookupTable
void test_String();
void test_integer();

int main()
{

 //create and test a lookup table with an integer key value and Customer datum
 //test_Customer();

 // Uncomment the following function calls when ready to test template class
LookupTable
 // create and test a a lookup table of type <int, String>
 //test_String();

 // Uncomment the following function calls when ready to test template class
LookupTable
 // create and test a a lookup table of type <int, int>
 test_integer();

  cout<<"\n\nProgram terminated successfully.\n\n";

  return 0;
}
```

```
template<class KeyType, class DatumType>
void print(LookupTable<KeyType, DatumType>& lt)
{
  if (lt.size() == 0)
    cout << "  Table is EMPTY.\n";
  for (lt.go_to_first(); lt.cursor_ok(); lt.step_fwd()) {
    cout << lt << endl;
  }
}

template<class KeyType, class DatumType>
void try_to_find(LookupTable<KeyType, DatumType>& lt, KeyType key)
{
  lt.find(key);
  if (lt.cursor_ok())
    cout << "\nFound key:"  << lt;
  else
    cout << "\nSorry, I couldn't find key: " << key << " in the table.\n";
}

void test_Customer()
  //creating a lookup table for customer objects.
  {
    // cout<<"\nCreating and testing Customers Lookup Table <not
template>-...\n";
    // LookupTable lt;

    // // Insert using new keys.
    // Customer a("Joe", "Morrison", "11 St. Calgary.", "(403)-1111-123333");
    // Customer b("Jack", "Lewis", "12 St. Calgary.", "(403)-1111-123334");
    // Customer c("Tim", "Hardy", "13 St. Calgary.", "(403)-1111-123335");
    // lt.insert(Pair (8002, a));
    // lt.insert(Pair (8004, c));
    // lt.insert(Pair (8001, b));

    // assert(lt.size() == 3);
    // lt.remove(8004);
    // assert(lt.size() == 2);
```

```
    // cout << "\nPrinting table after inserting 3 new keys and 1
removal...\n";
    // print(lt);

    // // Pretend that a user is trying to look up customers info.

    // cout << "\nLet's look up some names ...\n";
    // try_to_find(lt, 8001);
    // try_to_find(lt, 8000);

    // // test Iterator
    // cout << "\nTesing and using  iterator ...\n";
    // LookupTable::Iterator it = lt.begin();
    // cout <<"\nThe first node contains: " <<*it <<endl;

    // while (!it) {
    //   cout <<++it << endl;
    // }

    // //test copying
    // lt.go_to_first();
    // lt.step_fwd();
    // LookupTable clt(lt);
    // assert(strcmp(clt.cursor_datum().getFname(),"Joe")==0);

    // cout << "\nTest copying: keys should be 8001, and 8002\n";
    // print(clt);
    // lt.remove(8002);

    // //Assignment operator check.
    // clt= lt;

    // cout << "\nTest assignment operator: key should be 8001\n";
    // print(clt);

    // //Wipe out the entries in the table.
    // lt.make_empty();
    // cout << "\nPrinting table for the last time: Table should be
```

```
empty...\n";
    // print(lt);


    // cout << "***----Finished tests on Customers Lookup Table <not
template>-----***\n";
    // cout << "PRESS RETURN TO CONTINUE.";
    // cin.get();

  }

// Uncomment and modify the following funciton when ready to test
LookupTable<int,Mystring>

void test_String()

  // creating lookuptable for Mystring objects
  {
    cout<<"\nCreating and testing LookupTable <int, Mystring> .....\n";
    LookupTable<int, Mystring> lt;

    // Insert using new keys.

    Mystring a("I am an ENEL-409 student.");
    Mystring b("C++ is a powerful language for engineers but it's not easy.");
    Mystring c ("Winter 2004");

    lt.insert(Pair<int, Mystring> (8002,a));
    lt.insert(Pair<int, Mystring> (8001,b));
    lt.insert(Pair<int, Mystring> (8004,c));

    //assert(lt.size() == 3);
    //lt.remove(8004);
    //assert(lt.size() == 2);
    cout << "\nPrinting table after inserting 3 new keys and  and 1
removal...\n";
    print(lt);
```

```
  // Pretend that a user is trying to look up customers info.

  cout << "\nLet's look up some names ...\n";
  try_to_find(lt, 8001);
  try_to_find(lt, 8000);
  // test Iterator
  LookupTable<int, Mystring>::Iterator it = lt.begin();
  cout <<"\nThe first node contains: " <<*it <<endl;

  while (!it) {
    cout <<++it << endl;
  }

  //test copying
  lt.go_to_first();
  lt.step_fwd();
  LookupTable  clt(lt);
  assert(strcmp(clt.cursor_datum().c_str(),"I am an ENEL-409 student.")==0);

  cout << "\nTest copying: keys should be 8001, and 8002\n";
  print(clt);
  lt.remove(8002);

  //Assignment operator check.
  clt= lt;

  cout << "\nTest assignment operator: key should be 8001\n";
  print(clt);

  // Wipe out the entries in the table.
  lt.make_empty();
  cout << "\nPrinting table for the last time: Table should be empty ...\n";
  print(lt);

  cout << "***----Finished Lab 4 tests on <int> <Mystring>-----***\n";
  cout << "PRESS RETURN TO CONTINUE.";
  cin.get();
}
```

```
// // Uncomment and modify the following funciton when ready to test
LookupTable<int,int>

void test_integer()

  //creating look table of integers

  {
    cout<<"\nCreating and testing LookupTable <int, int> .....\n";
    LookupTable<int, int> lt;

    // Insert using new keys.
    lt.insert(Pair<int, int>(8002,9999));
    lt.insert(Pair<int, int>(8001,8888));
    lt.insert(Pair<int, int>(8004,8888));
    assert(lt.size() == 3);
    lt.remove(8004);
    assert(lt.size() == 2);
    cout << "\nPrinting table after inserting 3 new keys and  and 1
removal...\n";
    print(lt);

    // Pretend that a user is trying to look up customers info.

    cout << "\nLet's look up some names ...\n";
    try_to_find(lt, 8001);
    try_to_find(lt, 8000);

    // test Iterator
    LookupTable<int, int>::Iterator it = lt.begin();

    while (!it) {
      cout <<++it << endl;
```

```
    }

    //test copying
    lt.go_to_first();
    lt.step_fwd();
    LookupTable clt(lt);
    assert(clt.cursor_datum()== 9999);

    cout << "\nTest copying: keys should be 8001, and 8002\n";
    print(clt);
    lt.remove(8002);

    //Assignment operator check.
    clt= lt;

    cout << "\nTest assignment operator: key should be 8001\n";
    print(clt);


    // Wipe out the entries in the table.
    lt.make_empty();
    cout << "\nPrinting table for the last time: Table should be empty ...\n";
    print(lt);

    cout << "***----Finished Lab 4 tests on <int> <int>-----***\n";

}
```