Names: Findlay Brown (30145677), Nimna Wijedasa (30146042)

Course Name: Principles of Software Design

Lab Section: B02

Course Code: ENSF 480

Assignment Number: Lab-2

Submission Date: 11/10/2023

Exercise A

```cpp
/*
 * File Name: circle.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */
#include "circle.h"

Circle::Circle(double x, double y, const char *name, double radius)
    : Shape(x, y, name)
{
    this->radius = radius;
}

Circle::Circle(double x, double y, double radius, const char *name)
    : Circle(x, y, name, radius)
{
}

Circle::Circle(Point origin, const char *name, double radius)
    : Shape(origin, name)
{
    this->radius = radius;
}

Circle::Circle(Point origin, double radius, const char *name)
    : Circle(origin, name, radius)
{
}

Circle::Circle(const Circle &source)
    : Shape(source)
{
    this->radius = source.radius;
```

```
}

Circle &Circle::operator=(const Circle &rhs)
{
    if (this != &rhs)
    {
        Shape::operator=(rhs);
        this->radius = rhs.radius;
    }
    return *this;
}

double Circle::getRadius() const
{
    return this->radius;
}

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

void Circle::display() const
{
    std::cout << "Circle Name:\t" << this->shapeName << std::endl;
    origin.display();
    std::cout << "Radius:\t\t" << this->radius << std::endl;
}

double Circle::area() const
{
    // A = pi*r^2
    return (M_PI * pow(this->radius, 2));
}

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

```
    // P = 2*pi*r
    return (2 * M_PI * this->radius);
}
```

```
/*
 * File Name: circle.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#pragma once

#include "shape.h"
#include <cmath>

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

public:
    // Constructor
    Circle(double x, double y, const char *name, double radius);

    // Constructor Overload 1
    Circle(double x, double y, double radius, const char *name);

    // Constructor Overload 2
    Circle(Point origin, const char *name, double radius);

    // Constructor Overload 3
    Circle(Point origin, double radius, const char *name);

    // Circle Copy constructor
```

```
    Circle(const Circle &source);

    // Circle Assignment operator
    Circle &operator=(const Circle &rhs);

    // radius getter
    double getRadius() const;

    // radius setter
    void setRadius(double radius);

    virtual void display() const;
    virtual double area() const;
    virtual double perimeter() const;
};
```

```
/*
 * File Name: main.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#include "graphicsWorld.h"
int main(int argc, char const *argv[])
{
    GraphicsWorld g;
    g.run();
    return 0;
}

// Compile with the following
// g++ -o exeAmain.exe main.cpp graphicsWorld.cpp rectangle.cpp square.cpp shape.cpp point.cpp circle.cpp
curveCut.cpp
```

```h
/*
 * File Name: graphicsWorld.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#ifndef _GRAPHICSWORLD_H
#define _GRAPHICSWORLD_H

#include "point.h"
#include "shape.h"
#include "circle.h"
#include "square.h"
#include "rectangle.h"
#include "curveCut.h"

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

#endif
```

```cpp
/*
 * File Name: graphicsWorld.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */
```

```cpp
#include "graphicsWorld.h"
#include <iostream>
using namespace std;

void GraphicsWorld::run()
{
#if 1
    // 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:" << endl;
    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:\tRECTANGLE A\n"
         << "X-coordinate:\t5\n"
         << "Y-coordinate:\t7\n"
         << "Side a:\t\t12\n"
         << "Side b:\t\t15\n"
         << "Area:\t\t180\n"
         << "Perimeter:\t54\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 rec3: " << endl;
    cout << "Rectangle Name:\tRECTANGLE A\n"
         << "X-coordinate:\t5\n"
         << "Y-coordinate:\t7\n"
         << "Side a:\t\t100\n"
         << "Side b:\t\t200\n"
         << "Area:\t\t20000\n"
         << "Perimeter:\t600\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;
```

```
    sh[0]->display();
    sh[1]->display();
    sh[2]->display();
    sh[3]->display();
#endif // end of block to test array of pointer and polymorphism
/* *************************ASSUME CODE SEGMENT FOR EXERCISE A IS HERE **********************/
#if 1
    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.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 << "\nthe 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();
    cout << "\nthe perimeter of " << sh[0]->getName() << " is: " << sh[0]->perimeter() << "\n";
    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() << "\n";
    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() << "\n";
    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();
    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
```

```
/*
 * File Name: square.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#pragma once

#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "shape.h"

using namespace std;

class Square : virtual public Shape
{
protected:
    double side_a;

public:
```

```
    // Constructor
    Square(double x, double y, const char *name, double side_a);

    // Constructor Overload 1
    Square(double x, double y, double side_a, const char *name);

    // Constructor Overload 2
    Square(Point p, const char *name, double side_a);

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

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

    // side_a getter
    double getSideA() const;

    // side_a setter
    void setSideA(double side_a);
    virtual void display() const;

    virtual double area() const;
    virtual double perimeter() const;
};
```

```
/*
 * File Name: square.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#include <iostream>
```

```cpp
#include <stdlib.h>
#include "square.h"

using namespace std;

Square::Square(double x, double y, const char *name, double side_a)
    : Shape(x, y, name)
{
    this->side_a = side_a;
}
Square::Square(double x, double y, double side_a, const char *name)
    : Square(x, y, name, side_a)
{
}
Square::Square(Point p, const char *name, double side_a)
    : Shape(p, name)
{
    this->side_a = side_a;
}
Square::Square(const Square &source)
    : Shape(source)
{
    this->side_a = source.side_a;
}

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

double Square::getSideA() const
{
```

```
    return this->side_a;
}

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

void Square::display() const
{
    std::cout << "Square Name:\t" << shapeName << std::endl;
    origin.display();
    std::cout << "Side-a:\t\t" << side_a << std::endl;
    std::cout << "Area:\t\t" << area() << std::endl;
    std::cout << "Perimeter:\t" << perimeter() << std::endl;
}

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

double Square::perimeter() const
{
    return (this->side_a * 4);
}
```

```
/*
 * File Name: shape.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */
```

```cpp
#pragma once

#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include "point.h"

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

public:
    // Constructor
    Shape(double x, double y, const char *name);

    // Constructor Overload 1
    Shape(Point origin, const char *name);

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

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

    // Shape Destructor
    virtual ~Shape();

    // origin getter
    const Point &getOrigin() const;

    // name getter
    const char *getName() const;

    // name setter
```

```
    void setName(const char *name);

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

    void move(double dx, double dy);

    virtual void display() const;
    virtual double area() const = 0;
    virtual double perimeter() const = 0;
};
```

```
/*
 * File Name: shape.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#include "shape.h"

using namespace std;

Shape::Shape(double x, double y, const char *name)
    : origin(x, y)
{
    shapeName = new char[strlen(name) + 1];
    strcpy(shapeName, name);
}

Shape::Shape(Point origin, const char *name)
    : origin(origin)
{
    shapeName = new char[strlen(name) + 1];
```

```
    strcpy(shapeName, name);
}

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

Shape &Shape::operator=(const Shape &rhs)
{
    if (this != &rhs)
    {
        delete[] this->shapeName;
        this->shapeName = new char[strlen(rhs.shapeName) + 1];
        strcpy(this->shapeName, rhs.shapeName);
        this->origin = rhs.origin;
    }
    return *this;
}

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

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

const char *Shape::getName() const
{
    return this->shapeName;
}
```

```
void Shape::setName(const char *name)
{
    this->shapeName = new char[strlen(name) + 1];
    strcpy(shapeName, name);
}

void Shape::display() const
{
    cout << "Shape Name:\t" << shapeName << endl;
    origin.display();
}

double Shape::distance(Shape &otherShape)
{
    return origin.distance(otherShape.getOrigin());
}

double Shape::distance(Shape &shape1, Shape &shape2)
{
    return Point::distance(shape1.getOrigin(), shape2.getOrigin());
}

void Shape::move(double dx, double dy)
{
    origin.move(dx, dy);
}
```

```
/*
 * File Name: rectangle.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
```

```
 */

#pragma once

#include "square.h"
using namespace std;

class Rectangle : virtual public Square
{
protected:
    double side_b;

public:
    // Constructor
    Rectangle(double x, double y, const char *name, double side_a, double side_b);

    // Constructor Overload 1
    Rectangle(double x, double y, double side_a, double side_b, const char *name);

    // Constructor Overload 2
    Rectangle(Point origin, double side_a, double side_b, const char *name);

    // Constructor Overload 3
    Rectangle(Point origin, const char *name, double side_a, double side_b);

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

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

    // side_b getter
    double getSideB() const;

    // side_b setter
    void setSideB(double side_b);
```

```
    virtual void display() const;
    virtual double area() const;
    virtual double perimeter() const;
};
```

```
/*
 * File Name: rectangle.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#include <iostream>
#include <stdlib.h>
#include "rectangle.h"

using namespace std;

Rectangle::Rectangle(double x, double y, const char *name, double side_a, double side_b)
    : Square(x, y, name, side_a), Shape(x, y, name)
{
    this->side_b = side_b;
}

Rectangle::Rectangle(double x, double y, double side_a, double side_b, const char *name)
    : Rectangle(x, y, name, side_a, side_b)
{
}

Rectangle::Rectangle(Point origin, double side_a, double side_b, const char *name)
    : Square(origin, name, side_a),
      Shape(origin, name)
{
    this->side_b = side_b;
```

```
}

Rectangle::Rectangle(Point origin, const char *name, double side_a, double side_b)
    : Square(origin, name, side_a),
      Shape(origin, name)
{
    this->side_b = side_b;
}

Rectangle::Rectangle(const Rectangle &source)
    : Square(source),
      Shape(source)
{
    this->side_b = source.side_b;
}

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

double Rectangle::getSideB() const
{
    return this->side_b;
}

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

void Rectangle::display() const
```

```
{
    std::cout << "Rectangle Name:\t" << shapeName << std::endl;
    origin.display();
    std::cout << "Side-a:\t\t" << this->side_a << std::endl;
    std::cout << "Side-b:\t\t" << this->side_b << std::endl;
    std::cout << "Area:\t\t" << area() << std::endl;
    std::cout << "Perimeter:\t" << perimeter() << std::endl;
}

double Rectangle::area() const
{
    return (this->side_a * this->side_b);
}

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

```
/*
 * File Name: point.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#pragma once

#include <assert.h>
#include <iostream>
#include <stdlib.h>

using namespace std;
class Point
```

```
{
    friend ostream &operator<<(ostream &os, Point &p);

private:
    double x;
    double y;
    const int ID;
    static int count;
    static int autoIDnum;

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

    // Constructor Overload 1
    Point(const Point &source);

    // Point Assignment operator
    Point &operator=(const Point &rhs);

    // Point Destructor
    ~Point();

    // x getter
    double getX() const;

    // x setter
    void setX(double x);

    // y getter
    double getY() const;

    // y setter
    void setY(double y);

    int getId() const;
```

```
    double distance(const Point &otherPoint) const;
    static double distance(const Point &point1, const Point &point2);
    static int counter();
    void display() const;
    void move(double dx, double dy);
};
```

```
/*
 * File Name: point.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

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

int Point::count = 1;
int Point::autoIDnum = 1001;

Point::Point(double x, double y)
    : ID(autoIDnum)
{
    count++;
    autoIDnum++;
    this->x = x;
    this->y = y;
}

Point::Point(const Point &source)
    : ID(autoIDnum)
{
    count++;
    autoIDnum++;
```

```
    this->x = source.x;
    this->y = source.y;
}

Point &Point::operator=(const Point &rhs)
{
    if (this != &rhs)
    {
        this->x = rhs.x;
        this->y = rhs.y;
    }
    return *this;
}

Point::~Point()
{
    count--;
}

double Point::getX() const
{
    return this->x;
}

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

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

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

```
}

int Point::getId() const
{
    return this->ID;
}

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

double Point::distance(const Point &point1, const Point &point2)
{
    double dx = point1.x - point2.x;
    double dy = point1.y - point2.y;
    return (std::sqrt(dx * dx + dy * dy));
}

double Point::distance(const Point &otherPoint) const
{
    return distance(*this, otherPoint);
}

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

void Point::move(double dx, double dy)
{
    this->x += dx;
    this->y += dy;
}

ostream &operator<<(ostream &os, Point &p)
```

```
{
    os << "X-coordinate: " << p.x << std::endl;
    return os << "Y-coordinate: " << p.y << std::endl;
}
```

```
/*
 * File Name: curveCut.h
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */
#pragma once

#include "circle.h"
#include "rectangle.h"

class CurveCut : public Circle, public Rectangle
{

public:
    // Constructor Overload 1
    CurveCut(double x, double y, const char *name,
            double side_a, double side_b, double radius);
    // Constructor Overload 2
    CurveCut(double x, double y, double side_a,
            double side_b, double radius, const char *name);

    // CurveCut Copy constructor
    CurveCut(const CurveCut &source);

    // CurveCut Assignment operator
    CurveCut &operator=(const CurveCut &rhs);

    void display() const;
    double area() const;
```

```
    double perimeter() const;
};
```

```
/*
 * File Name: curveCut.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */
#include "curveCut.h"

CurveCut::CurveCut(double x, double y, const char *name, double side_a, double side_b, double radius)
    : Circle(x, y, name, radius),
      Rectangle(x, y, name, side_a, side_b),
      Square(x, y, name, side_a),
      Shape(x, y, name)
{
    assert(radius <= (side_a > side_b ? side_b : side_a));
}

CurveCut::CurveCut(double x, double y, double side_a, double side_b, double radius, const char *name)
    : CurveCut(x, y, name, side_a, side_b, radius)
{
}

CurveCut::CurveCut(const CurveCut &source)
    : Circle(source),
      Rectangle(source),
      Square(source),
      Shape(source)
{
}

CurveCut &CurveCut::operator=(const CurveCut &rhs)
```

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

void CurveCut::display() const
{
    std::cout << "CurveCut Name:\t" << this->shapeName << std::endl;
    origin.display();
    std::cout << "Width:\t\t" << this->side_a << std::endl;
    std::cout << "Length:\t\t" << this->side_b << std::endl;
    std::cout << "Radius of the cut: " << this->radius << std::endl;
}

double CurveCut::area() const
{
    // A = Arectangle - (1/4)Acircle
    return (Rectangle::area() - (0.25 * Circle::area()));
}

double CurveCut::perimeter() const
{
    // P = (1/4)Pcircle + Prectangle - 2Rcircle
    // Arectangle - (1/4)Acircle
    return ((0.25 * Circle::perimeter()) + Rectangle::perimeter() - 2 * this->radius);
}
```

```
➜  ExerciseA git:(main) ✗ ./exeAmain.out

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.
```

```
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 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 assignment operator.

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

Testing Functions in class Circle:
Circle Name:    CIRCLE C
X-coordinate:   3
Y-coordinate:   5
Radius:         9
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: 40.1372

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 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 Ais: 120000
the perimeter of SQUARE - S is: 1400
Circle Name:    CIRCLE C
X-coordinate:   3
Y-coordinate:   5
Radius:         9

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: 40.1372
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
➜  ExerciseA git:(main) ✗
```

Exercise B

```cpp
/*
 * File Name: iterator.cpp
 * Assignment: Lab 3 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

#include <iostream>
#include <assert.h>
#include <cstring>
#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);
    // PRIMISES: 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 &a, T &b); // 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 <>
void Vector<const char *>::ascending_sort()
{
  for (int i = 0; i < size - 1; i++)
    for (int j = i + 1; j < size; j++)
      if (strcmp(array[i], array[j]) > 0)
        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);
}

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

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

template <class T>
T Vector<T>::VectIter::operator++()
{
  this->index++;
```

```
  if (this->index > this->v->size - 1)
  {
    this->index = 0;
  }
  T returnValue = this->v->operator[](this->index);
  return returnValue;
}

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

  return returnValue;
}

template <class T>
T Vector<T>::VectIter::operator--()
{

  this->index--;
  if (this->index < 0)
  {
    this->index = this->v->size - 1;
  }
  T returnValue = this->v->operator[](this->index);
  return returnValue;
}

template <class T>
T Vector<T>::VectIter::operator--(int)
{
```

```
  T returnValue = this->v->operator[](this->index);
  this->index--;
  if (this->index < 0)
  {
    this->index = this->v->size - 1;
  }

  return returnValue;
}

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 0
  cout << "\nTesting an <int> Vector: " << endl;

  cout << "\n\nTesting Sort & Postfix ++";
  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--;
#if 1
  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 << "\nProgram Terminated Successfully." << endl;

 return 0;
}
```

```
/*
 * File Name: mystring2.cpp
 * Assignment: Lab 3 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
```

```
 * Submission Date: Oct 11, 2023
 */

#include "mystring2.h"
#include <string.h>
#include <assert.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 this->lengthM;
}

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

  return this->charsM[pos];
}

const char *Mystring::c_str() const
{
  return this->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;
  }

  this->charsM[pos] = c;
}

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

  return *this;
}

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

ostream &operator<<(ostream &os, Mystring &s)
{
  return os << s.charsM;
}
```

```
ostream &operator<<(ostream &os, const Mystring &s)
{
  return os << s.charsM;
}

char &Mystring::operator[](int index)
{
  return charsM[index];
}

char &Mystring::operator[](int index) const
{
  return charsM[index];
}

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);
```

```
}

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

```
/*
 * File Name: mystring2.h
 * Assignment: Lab 3 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

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

#ifndef MYSTRING_H
#define MYSTRING_H

class Mystring
{
  friend ostream &operator<<(ostream &os, const Mystring &s);
  friend ostream &operator<<(ostream &os, Mystring &s);

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

bool operator>(const Mystring &rhs) const;

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.

char &operator[](int index);

char &operator[](int index) const;

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.

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
```

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


Testing Sort & Postfix ++
-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
Apple
Orange
Pear
Program Terminated Successfully.
➜  ExerciseB git:(main) ✗
```

Exercise C

```
/*
 * File Name: lookupTable.h
 * Assignment: Lab 3 Exercise C
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
```

```
 * Submission Date: Oct 11, 2023
 */

#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 LT_Key;
typedef Customer LT_Datum;

template <class Key, class Data>
class LookupTable;

template <class Key, class Data>
struct Pair
{
  Pair(Key keyA, Data datumA) : key(keyA), datum(datumA)
  {
  }
```

```
  Key key;
  Data datum;
};

template <class Key, class Data>
class LT_Node
{
  friend class LookupTable<Key, Data>;

private:
  Pair<Key, Data> pairM;
  LT_Node<Key, Data> *nextM;

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

template <class Key, class Data>
class LookupTable
{
public:
  // Nested class
  class Iterator
  {
    friend class LookupTable;
    LookupTable *LT;
    //    LT_Node* cursor;

  public:
    Iterator() : LT(0) {}
    Iterator(LookupTable &x) : LT(&x) {}
    const Data &operator*();
    const Data &operator++();
    const Data &operator++(int);
    int operator!();

    void step_fwd()
```

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

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

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

void insert(const Pair<Key, Data> &pairA);
// 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.

  template <class K, class D>
  friend ostream &operator<<(ostream &os, const LookupTable<K, D> &lt);

private:
  int sizeM;
  LT_Node<Key, Data> *headM;
  LT_Node<Key, Data> *cursorM;

  void destroy();
```

```
  // Deallocate all nodes, set headM to zero.

  void copy(const LookupTable &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 Key, class Data>
LookupTable<Key, Data> &LookupTable<Key, Data>::begin()
{
  cursorM = headM;
  return *this;
}

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

template <class Key, class Data>
LookupTable<Key, Data>::LookupTable()
    : sizeM(0), headM(0), cursorM(0)
{
}

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

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

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

template <class Key, class Data>
LookupTable<Key, Data>::~LookupTable()
{
  destroy();
}

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

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

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

template <class Key, class Data>
const Data &LookupTable<Key, Data>::cursor_datum() const
{
  assert(cursor_ok());
```

```
  return cursorM->pairM.datum;
}

template <class Key, class Data>
void LookupTable<Key, Data>::insert(const Pair<Key, Data> &pairA)
{
  // Add new node at head?
  if (headM == 0 || pairA.key < headM->pairM.key)
  {
    headM = new LT_Node<Key, Data>(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<Key, Data> *before = headM;
    LT_Node<Key, Data> *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<Key, Data>(pairA, before->nextM);
```

```
    sizeM++;
   }
  }
}

template <class Key, class Data>
void LookupTable<Key, Data>::remove(const Key &keyA)
{

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

  LT_Node<Key, Data> *doomed_node = 0;
  if (keyA == headM->pairM.key)
  {
    doomed_node = headM;
    headM = headM->nextM;
    sizeM--;
  }
  else
  {
    LT_Node<Key, Data> *before = headM;
    LT_Node<Key, Data> *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 Key, class Data>
void LookupTable<Key, Data>::find(const Key &keyA)
{
  LT_Node<Key, Data> *ptr = headM;
  while (ptr != NULL && ptr->pairM.key != keyA)
  {
    ptr = ptr->nextM;
  }

  cursorM = ptr;
}

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

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

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

template <class Key, class Data>
void LookupTable<Key, Data>::destroy()
```

```
{

  LT_Node<Key, Data> *ptr = headM;
  while (ptr != NULL)
  {
    headM = headM->nextM;
    delete ptr;
    ptr = headM;
  }
  cursorM = NULL;
  sizeM = 0;
}

template <class Key, class Data>
void LookupTable<Key, Data>::copy(const LookupTable &source)
{

  headM = 0;
  cursorM = 0;

  if (source.headM == 0)
    return;

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

template <class Key, class Data>
ostream &operator<<(ostream &os, const LookupTable<Key, Data> &lt)
{
  if (lt.cursor_ok())
  {
    os << lt.cursor_key() << "  " << lt.cursor_datum();
```

```
  }
  else
  {
    os << "Not Found.";
  }

  return os;
}

// Iterator functions
template <class Key, class Data>
const Data &LookupTable<Key, Data>::Iterator::operator*()
{
  assert(LT->cursor_ok());
  return LT->cursor_datum();
}

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

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

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

template <class Key, class Data>
int LookupTable<Key, Data>::Iterator::operator!()
```

```
{
  return (LT->cursor_ok());
}
```

```
/*
 * File Name: mainLab3ExC.cpp
 * Assignment: Lab 3 Exercise A
 * Lab Section: B02
 * Completed by: Findlay Brown, Nimna Wijedasa
 * Submission Date: Oct 11, 2023
 */

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

#include <assert.h>
#include <iostream>
#include "lookupTable.h"
#include "customer.h"
#include "mystring2.h"
#include <cstring>
using namespace std;

template <class Key, class Data>
void print(LookupTable<Key,Data>& lt);


template <class Key, class Data>
void try_to_find(LookupTable<Key,Data>& lt, int 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 Key, class Data>
void print(LookupTable<Key,Data>& 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 Key, class Data>
void try_to_find(LookupTable<Key,Data>& lt, int 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<int, Customer> 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<int,Customer>(8002, a));
    lt.insert(Pair<int,Customer> (8004,c));
    lt.insert(Pair<int,Customer> (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<int, Customer>::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<int,Customer>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<int, Mystring>   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<int, int> 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";


}
```

```
➜  ExerciseC git:(main) ✗ g++ -o exeCmain.out customer.cpp mainLab3ExC.cpp mystring2.cpp
➜  ExerciseC git:(main) ✗ ./exeCmain.out

  Creating and testing Customers Lookup Table <not template>-...

  Printing table after inserting 3 new keys and 1 removal...
  8001  Name: Jack Lewis. Address: 12 St. Calgary.. Phone:: (403)-1111-123334
  8002  Name: Joe Morrison. Address: 11 St. Calgary.. Phone:: (403)-1111-123333

  Let's look up some names ...

  Found key:8001  Name: Jack Lewis. Address: 12 St. Calgary.. Phone:: (403)-1111-123334
  Sorry, I couldn't find key: 8000 in the table.

  Tesing and using  iterator ...

  The first node contains: Name: Jack Lewis. Address: 12 St. Calgary.. Phone:: (403)-1111-123334
  Name: Jack Lewis. Address: 12 St. Calgary.. Phone:: (403)-1111-123334
  Name: Joe Morrison. Address: 11 St. Calgary.. Phone:: (403)-1111-123333

  Test copying: keys should be 8001, and 8002
  8001  Name: Jack Lewis. Address: 12 St. Calgary.. Phone:: (403)-1111-123334
  8002  Name: Joe Morrison. Address: 11 St. Calgary.. Phone:: (403)-1111-123333

  Test assignment operator: key should be 8001
  8001  Name: Jack Lewis. Address: 12 St. Calgary.. Phone:: (403)-1111-123334

  Printing table for the last time: Table should be empty...
    Table is EMPTY.
  ***----Finished tests on Customers Lookup Table <not template>-----***
  PRESS RETURN TO CONTINUE.
```

```
Creating and testing LookupTable <int, Mystring> .....

Printing table after inserting 3 new keys and  and 1 removal...
8001  C++ is a powerful language for engineers but it's not easy.
8002  I am an ENEL-409 student.

Let's look up some names ...

Found key:8001  C++ is a powerful language for engineers but it's not easy.
Sorry, I couldn't find key: 8000 in the table.

The first node contains: C++ is a powerful language for engineers but it's not easy.
C++ is a powerful language for engineers but it's not easy.
I am an ENEL-409 student.

Test copying: keys should be 8001, and 8002
8001  C++ is a powerful language for engineers but it's not easy.
8002  I am an ENEL-409 student.

Test assignment operator: key should be 8001
8001  C++ is a powerful language for engineers but it's not easy.

Printing table for the last time: Table should be empty ...
  Table is EMPTY.
***----Finished Lab 4 tests on <int> <Mystring>-----***
PRESS RETURN TO CONTINUE.
```

```
Creating and testing LookupTable <int, int> .....

Printing table after inserting 3 new keys and  and 1 removal...
8001  8888
8002  9999

Let's look up some names ...

Found key:8001  8888
Sorry, I couldn't find key: 8000 in the table.
8888
9999

Test copying: keys should be 8001, and 8002
8001  8888
8002  9999

Test assignment operator: key should be 8001
8001  8888

Printing table for the last time: Table should be empty ...
  Table is EMPTY.
***----Finished Lab 4 tests on <int> <int>-----***


Program terminated successfully.

➜  ExerciseC git:(main) ✗
```