Course: ENSF614 – Fall 2025

Lab #: Lab 3

Instructor: Mahmood Moussavi

Students Name: John Zhou, Muhammad Ibrahim Khan

Submission Date: Oct 1st, 2025  
  
  
  
  
  
  
I have been keeping all the files in github. I hope by providing this github link will help you a little bit.  
https://github.com/JZ-Zhou-UofC/ENSF-614-assignment-repo

Exercise A

/\*

\*

\*

File Name: circle.cpp

Assignment: Lab 3 Exercise A

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: oct 1st, 2025

\*/

#include <iomanip>

#include <iostream>

#include "circle.h"

using namespace std;

const double Circle::pi = 3.141592653589793;

// Constructor

Circle::Circle(double x, double y, double r, const char \*name)

: Shape(Point(x, y), name), radius(r) {}

double Circle::get\_radius() const { return radius; }

void Circle::set\_radius(double r) { radius = r; }

double Circle::area() const { return radius\* radius\*pi; }

double Circle::perimeter() const { return 2\*pi \* radius; }

void Circle::display() const {

cout << "Circle Name: " << (getName()) << "\n";

cout << fixed << setprecision(2);

cout << "X-coordinate: " << getOrigin().getx() << "\n";

cout << "Y-coordinate: " << getOrigin().gety() << "\n";

cout << "Radius: " << radius << "\n";

cout << "Area: " << area() << "\n";

cout << "Perimeter: " << perimeter() << "\n";

}

*/\**

*\**

*\**

*File Name: circle.h*

*Assignment: Lab 3 Exercise A*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: oct 1st, 2025*

*\*/*

#ifndef *CIRCLE\_H*

#define *CIRCLE\_H*

#include "shape.h"

class *Circle* : virtual public *Shape* {

 private:

  double radius;

 public:

  static const double pi;

*Circle*(double *x*, double *y*, double *radius*, const char\* *name*);

*Circle*(double *r*, const *Point*& *pt*, const char\* *n*) : *Shape*(pt, n), *radius*(r) {}

  double *get\_radius*() const;

  void *set\_radius*(double *r*);

  virtual double *area*() const;

  virtual double *perimeter*() const;

  virtual void *display*() const;

};

#endif

/\*

\*

\*

File Name: curveCut.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "curveCut.h"

#include <iomanip>

#include <iostream>

using namespace std;

CurveCut::CurveCut(double x, double y, double width, double length,

double radius, const char\* name)

: Shape(Point(x, y), name),

Rectangle(x, y, width, length, name),

Circle(x, y, radius, name) {

if (radius > std::min(width, length)) {

cerr << "Error: radius must be <= min(width, length)" << endl;

exit(EXIT\_FAILURE);

}

}

double CurveCut::area() const {

return Rectangle::area() - 0.25 \* Circle::area();

}

double CurveCut::perimeter() const {

return Rectangle::perimeter() - 2 \* Circle::get\_radius() +

0.5 \* Circle::perimeter();

}

void CurveCut::display() const {

cout << "CurveCut Name: " << getName() << "\n";

cout << "X-coordinate: " << getOrigin().getx() << "\n";

cout << "Y-coordinate: " << getOrigin().gety() << "\n";

cout << "Width: " << get\_side\_a() << "\n";

cout << "Length: " << get\_side\_b() << "\n";

cout << "Radius of the cut: " << Circle::get\_radius() << "\n";

}

*/\**

*\**

*\**

*File Name: rectangle.cpp*

*Assignment: Lab 2 Exercise B*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: Sept 24, 2025*

*\*/*

#ifndef *CURVECUT\_h*

#define *CURVECUT\_h*

#include "circle.h"

#include "rectangle.h"

class *CurveCut* : public *Rectangle*, *Circle* {

 public:

*CurveCut*(double *x*, double *y*, double *side\_a*, double *side\_b*, double *radius*,

           const char\* *name*);

  virtual double *area*() const override;

  virtual double *perimeter*() const override;

  virtual void *display*() const override;

};

#endif

/\*

\*

\*

File Name: graphicsWorld.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "graphicsWorld.h"

#include <iostream>

#include "circle.h"

#include "curveCut.h"

#include "point.h"

#include "rectangle.h"

#include "square.h"

using namespace std;

void GraphicsWorld::run() {

cout << "----------------------------------------------------\n";

cout << "Author: John Zhou\n";

cout << "----------------------------------------------------\n\n";

#if 1 // Change 0 to 1 to test Point

cout << "\nTesting class Point:" << endl;

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)

<< "\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 rectangle a and b is: " << d << endl;

Rectangle rec1 = a; // copy constructor

rec1.display();

cout << "\nTesting assignment operator in class Rectangle:" << endl;

Rectangle rec2(3, 4, 11, 7, "RECTANGLE rec2");

rec2.display();

rec2 = a; // assignment operator

a.set\_side\_b(200);

a.set\_side\_a(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.set\_side\_b(300);

a.set\_side\_a(400);

cout << "\nExpected to display the following values for object rec3:" << 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 copy constructor.\n"

<< endl;

rec3.display();

#endif // end of block to test Rectangle

#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;

CurveCut rc(6, 5, 10, 12, 9, "CurveCut rc");

rc.display();

cout << "the area of " << rc.getName() << " is: " << rc.area();

cout << "the perimeter of " << rc.getName() << " is: " << rc.perimeter();

d = rc.distance(c);

cout << "\nThe distance between rc and c is: " << d;

// 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();

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();

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();

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();

// end of block to test CurveCut

#endif // end of block to test array of pointer and polymorphism

}

*/\**

*\**

*\**

*File Name: graphicsWorld.h*

*Assignment: Lab 2 Exercise B*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: Sept 24, 2025*

*\*/*

#ifndef *GRAPHICSWORLD\_H*

#define *GRAPHICSWORLD\_H*

class *GraphicsWorld* {

public:

    void *run*();

};

#endif

/\*

\*

\*

File Name: Lab2\_exBmain.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "graphicsWorld.h"

int main() {

GraphicsWorld gw;

gw.run();

return 0;

}

/\*

\*

\*

File Name: point.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "point.h"

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

int Point::idCounter = 1000;

// Constructor

Point::Point(double x\_val, double y\_val) : x(x\_val), y(y\_val) {

idCounter ++;

id=idCounter;

}

// Getters

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

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

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

// Setters

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

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

void Point::display() const {

cout << fixed << setprecision(2);

cout << "X-coordinate: " << x << "\n";

cout << "Y-coordinate: " << y << "\n";

}

int Point::counter() {

return idCounter -1000;

}

double Point::distance(const Point& other) const {

double dx = x - other.x;

double dy = y - other.y;

return sqrt(dx \* dx + dy \* dy);

}

// Static distance (between any two 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);

}

*/\**

*\**

*\**

*File Name: point.h*

*Assignment: Lab 2 Exercise B*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: Sept 24, 2025*

*\*/*

#ifndef *POINT\_H*

#define *POINT\_H*

class *Point* {

private:

    double x;

    double y;

    int id;

    static int idCounter; *// Static member to track all ID*

public:

*Point*(double *x\_val*, double *y\_val*);

*// Getters*

    double *getx*() const;

    double *gety*() const;

    int *getId*() const;

*// Setters*

    void *setx*(double *x\_val*);

    void *sety*(double *y\_val*);

    void *display*() const;

*//static to access the static member*

    static int *counter*();

    double *distance*(const *Point*& *other*) const;

*//This can be static because this does not require an object of point*

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

};

#endif

/\*

\*

\*

File Name: rectangle.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "rectangle.h"

#include <iostream>

#include <iomanip>

using namespace std;

Rectangle::Rectangle(double x, double y, double side\_a, double side\_b, const char \*name)

: Shape(Point(x, y), name),Square(x, y, side\_a, name), side\_b(side\_b) {}

double Rectangle::get\_side\_b() const

{

return side\_b;

}

void Rectangle::set\_side\_b(double side)

{

side\_b = side;

}

double Rectangle::area() const

{

return get\_side\_a() \* side\_b;

}

double Rectangle::perimeter() const

{

return 2 \* (get\_side\_a() + side\_b);

}

void Rectangle::display() const

{

cout << "Rectangle Name: " << (getName()) << "\n";

cout << fixed << setprecision(2);

cout << "X-coordinate: " << getOrigin().getx() << "\n";

cout << "Y-coordinate: " << getOrigin().gety() << "\n";

cout << "Side a: " << get\_side\_a() << "\n";

cout << "Side b: " << side\_b << "\n";

cout << "Area: " << area() << "\n";

cout << "Perimeter: " << perimeter() << "\n";

}

*/\**

*\**

*\**

*File Name: rectangle.cpp*

*Assignment: Lab 2 Exercise B*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: Sept 24, 2025*

*\*/*

#ifndef *RECTANGLE\_H*

#define *RECTANGLE\_H*

#include "square.h"

class *Rectangle* :  public *Square*

{

private:

    double side\_b;

public:

*Rectangle*(double *x*, double *y*, double *side\_a*, double *side\_b*, const char \**name*);

    double *get\_side\_b*() const;

    void *set\_side\_b*(double *side*);

    virtual double *area*() const override;

    virtual double *perimeter*() const override;

    virtual void *display*() const override;

};

#endif

/\*

\*

\*

File Name: shape.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "shape.h"

#include <iostream>

#include <cstring>

#include <cmath>

#include <iomanip>

using namespace std;

// Constructor

Shape::Shape(const Point& pt, const char\* name) : origin(pt) {

shapeName = new char[strlen(name) + 1];

strcpy(shapeName, name);

}

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) {

origin = other.origin;

delete[] shapeName;

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;

}

void Shape::move(double dx, double dy) {

origin.setx(origin.getx() + dx);

origin.sety(origin.gety() + dy);

}

void Shape::display() const {

cout << "Shape Name: " << (shapeName ? shapeName : "Unnamed") << "\n";

cout << fixed << setprecision(2);

cout << "X-coordinate: " << origin.getx() << "\n";

cout << "Y-coordinate: " << origin.gety() << "\n";

}

double Shape::distance(Shape& other) {

return origin.distance(other.getOrigin());

}

double Shape::distance(Shape& s1, Shape& s2) {

return Point::distance(s1.getOrigin(), s2.getOrigin());

}

*/\**

*\**

*\**

*File Name: shape.h*

*Assignment: Lab 2 Exercise B*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: Sept 24, 2025*

*\*/*

#ifndef *SHAPE\_H*

#define *SHAPE\_H*

#include "point.h"

class *Shape* {

 protected:

*Point* origin;

  char\* shapeName;

 public:

*// shapeName is on heap*

*// Constructor*

*Shape*(const *Point*& *pt*, const char\* *name*);

*// Copy constructor*

*Shape*(const *Shape*& *other*);

*// Assignment operator*

*Shape*& *operator=*(const *Shape*& *other*);

  virtual *~Shape*();

*// Getters*

  const *Point*& *getOrigin*() const;

  const char\* *getName*() const;

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

  virtual void *display*() const;

  virtual double *area*() const = 0;

  virtual double *perimeter*() const = 0;

  double *distance*(*Shape*& *other*);

*// Static function, does not require shape object*

  static double *distance*(*Shape*& *s1*, *Shape*& *s2*);

};

#endif

/\*

\*

\*

File Name: square.cpp

Assignment: Lab 2 Exercise B

\* Completed by: John Zhou, Ibrahim Khan

\* Submission Date: Sept 24, 2025

\*/

#include "square.h"

#include <iomanip>

#include <iostream>

using namespace std;

// Constructor

Square::Square(double x, double y, double side, const char\* name)

: Shape(Point(x, y), name), side\_a(side) {}

double Square::get\_side\_a() const { return side\_a; }

void Square::set\_side\_a(double side) { side\_a = side; }

double Square::area() const { return side\_a \* side\_a; }

double Square::perimeter() const { return 4 \* side\_a; }

void Square::display() const {

cout << "Square Name: " << (getName()) << "\n";

cout << fixed << setprecision(2);

cout << "X-coordinate: " << getOrigin().getx() << "\n";

cout << "Y-coordinate: " << getOrigin().gety() << "\n";

cout << "Side a: " << side\_a << "\n";

cout << "Area: " << area() << "\n";

cout << "Perimeter: " << perimeter() << "\n";

}

*/\**

*\**

*\**

*File Name: square.h*

*Assignment: Lab 2 Exercise B*

*\*  Completed by: John Zhou, Ibrahim Khan*

*\*  Submission Date: Sept 24, 2025*

*\*/*

#ifndef *SQUARE\_H*

#define *SQUARE\_H*

#include "shape.h"

class *Square* : virtual public *Shape* {

 private:

  double side\_a;

 public:

*// Constructor*

*Square*(double *x*, double *y*, double *side*, const char \**name*);

  double *get\_side\_a*() const;

  void *set\_side\_a*(double *side*);

  virtual double *area*() const;

  virtual double *perimeter*() const;

  virtual void *display*() const;

};

#endif

Screenshot:

A screen shot of a computer

AI-generated content may be incorrect.

Exercise C

// LookupTable.h

// ENSF 614 - Lab 3, Ex C (Templated Version)

#ifndef LOOKUPTABLE\_H

#define LOOKUPTABLE\_H

#include <cassert>

#include <iostream>

using namespace std;

// Forward declaration

template <typename Key, typename Datum>

class LookupTable;

template <typename Key, typename Datum>

struct Pair {

Pair(Key keyA, Datum datumA) : key(keyA), datum(datumA) {}

Key key;

Datum datum;

};

// Node template

template <typename Key, typename Datum>

class LT\_Node {

friend class LookupTable<Key, Datum>;

private:

Pair<Key, Datum> pairM;

LT\_Node\* nextM;

LT\_Node(const Pair<Key, Datum>& pairA, LT\_Node\* nextA);

// LT\_Node(const Pair<Key, Datum>& pairA, LT\_Node\* nextA): pairM(pairA), nextM(nextA) {};

};

// LookupTable template

template <typename Key, typename Datum>

class LookupTable {

public:

// Nested Iterator

class Iterator {

friend class LookupTable<Key, Datum>;

LookupTable<Key, Datum>\* LT;

public:

Iterator() : LT(0) {}

Iterator(LookupTable<Key, Datum>& x) : LT(&x) {}

const Datum& operator\*();

const Datum& operator++();

const Datum& 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;

int cursor\_ok() const;

const Key& cursor\_key() const;

const Datum& cursor\_datum() const;

void insert(const Pair<Key, Datum>& pariA);

void remove(const Key& keyA);

void find(const Key& keyA);

void go\_to\_first();

void step\_fwd();

void make\_empty();

friend ostream& operator<<(ostream& os, const LookupTable<Key, Datum>& lt) {

if (lt.cursor\_ok())

os << lt.cursor\_key() << " " << lt.cursor\_datum();

else

os << "Not Found.";

return os;

}

private:

int sizeM;

LT\_Node<Key, Datum>\* headM;

LT\_Node<Key, Datum>\* cursorM;

void destroy();

void copy(const LookupTable& source);

};

// implementation

template <typename Key, typename Datum>

LT\_Node<Key, Datum>::LT\_Node(const Pair<Key, Datum>& pairA, LT\_Node\* nextA)

: pairM(pairA), nextM(nextA) {}

template <typename Key, typename Datum>

LookupTable<Key, Datum>& LookupTable<Key, Datum>::begin() {

cursorM = headM;

return \*this;

}

template <typename Key, typename Datum>

LookupTable<Key, Datum>::LookupTable() : sizeM(0), headM(0), cursorM(0) {}

template <typename Key, typename Datum>

LookupTable<Key, Datum>::LookupTable(const LookupTable& source) {

copy(source);

}

template <typename Key, typename Datum>

LookupTable<Key, Datum>& LookupTable<Key, Datum>::operator=(

const LookupTable& rhs) {

if (this != &rhs) {

destroy();

copy(rhs);

}

return \*this;

}

template <typename Key, typename Datum>

LookupTable<Key, Datum>::~LookupTable() {

destroy();

}

template <typename Key, typename Datum>

int LookupTable<Key, Datum>::size() const {

return sizeM;

}

template <typename Key, typename Datum>

int LookupTable<Key, Datum>::cursor\_ok() const {

return cursorM != 0;

}

template <typename Key, typename Datum>

const Key& LookupTable<Key, Datum>::cursor\_key() const {

assert(cursor\_ok());

return cursorM->pairM.key;

}

template <typename Key, typename Datum>

const Datum& LookupTable<Key, Datum>::cursor\_datum() const {

assert(cursor\_ok());

return cursorM->pairM.datum;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::insert(const Pair<Key, Datum>& pairA) {

if (headM == 0 || pairA.key < headM->pairM.key) {

headM = new LT\_Node<Key, Datum>(pairA, headM);

sizeM++;

} else if (pairA.key == headM->pairM.key)

headM->pairM.datum = pairA.datum;

else {

LT\_Node<Key, Datum>\* before = headM;

LT\_Node<Key, Datum>\* 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, Datum>(pairA, before->nextM);

sizeM++;

}

}

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::remove(const Key& keyA) {

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

LT\_Node<Key, Datum>\* doomed\_node = 0;

if (keyA == headM->pairM.key) {

doomed\_node = headM;

headM = headM->nextM;

sizeM--;

} else {

LT\_Node<Key, Datum>\* before = headM;

LT\_Node<Key, Datum>\* 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;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::find(const Key& keyA) {

LT\_Node<Key, Datum>\* ptr = headM;

while (ptr != NULL && ptr->pairM.key != keyA) {

ptr = ptr->nextM;

}

cursorM = ptr;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::go\_to\_first() {

cursorM = headM;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::step\_fwd() {

assert(cursor\_ok());

cursorM = cursorM->nextM;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::make\_empty() {

destroy();

sizeM = 0;

cursorM = 0;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::destroy() {

LT\_Node<Key, Datum>\* ptr = headM;

while (ptr != NULL) {

headM = headM->nextM;

delete ptr;

ptr = headM;

}

cursorM = NULL;

sizeM = 0;

}

template <typename Key, typename Datum>

void LookupTable<Key, Datum>::copy(const LookupTable& source) {

headM = 0;

cursorM = 0;

if (source.headM == 0) return;

for (LT\_Node<Key, Datum>\* p = source.headM; p != 0; p = p->nextM) {

insert(Pair<Key, Datum>(p->pairM.key, p->pairM.datum));

if (source.cursorM == p) find(p->pairM.key);

}

}

// ------------ Iterator Implementation ---------------

template <typename Key, typename Datum>

const Datum& LookupTable<Key, Datum>::Iterator::operator\*() {

assert(LT->cursor\_ok());

return LT->cursor\_datum();

}

template <typename Key, typename Datum>

const Datum& LookupTable<Key, Datum>::Iterator::operator++() {

assert(LT->cursor\_ok());

const Datum& x = LT->cursor\_datum();

LT->step\_fwd();

return x;

}

template <typename Key, typename Datum>

const Datum& LookupTable<Key, Datum>::Iterator::operator++(int) {

assert(LT->cursor\_ok());

LT->step\_fwd();

return LT->cursor\_datum();

}

template <typename Key, typename Datum>

int LookupTable<Key, Datum>::Iterator::operator!() {

return (LT->cursor\_ok());

}

#endif

// ENSF 614 - 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 <typename K, typename V>

void print(LookupTable<K, V>& 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 <typename K, typename V>

void try\_to\_find(LookupTable<K, V>& lt, K 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";

}

// TEST

void test\_Customer()

{

cout<<"\nCreating and testing Customers Lookup Table <int, Customer>...\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 << "\nTesting and using iterator ...\n";

typename 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 <int, Customer>-----\*\*\*\n";

cout << "PRESS RETURN TO CONTINUE.";

cin.get();

}

void test\_String()

{

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));

cout << "\nPrinting table after inserting 3 new keys...\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

typename 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();

}

void test\_integer()

{

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 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

typename 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";

cout << "PRESS RETURN TO CONTINUE.";

cin.get();

}

// ===================== MAIN =====================

int main()

{

test\_Customer(); // LookupTable<int, Customer>

test\_String(); // LookupTable<int, Mystring>

test\_integer(); // LookupTable<int, int>

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

return 0;

}

Screenshot

A screenshot of a computer program

AI-generated content may be incorrect.