Kyler Ferrell-Clegg | Lokesh Surya Paluri | Sai Charan Kadari
CIS4930
Group 2

Point2DImpl

Convention to follow:

((a,b),(c,d),(e,f),(g,h))

Point2DImpl-Function Definitions

Point2DImpl();

The first constructor creates a Point2D*Impl* object but, only defines our pointer to our struct and doesn't store any point information.

Point2DImpl(std::vector<RGPPoint2D> pointVector);

The second constructor creates a Point2DImpl object based on a passed in vector of RGP points.

Point2DImpl(std::string listOfPoint2DString);

The third constructor creates a Point2DImpl object based on a passed in string of points.

Point2DImpl(std::ifstream& file);

The final constructor creates a Point2DImpl object based on a passed in file of points.

~Point2DImpl();

The destructor clears any data that is stored in our vector of points or our bounding box vector.

std::string getPointString();

Returns a string containing all of the points in our vector of points.

void printAllPoints();

Prints out every stored point in our vector of points.

bool isEmptyPoint();

Returns true if vector of points is empty, false otherwise.

bool isValidPoint();

Returns true if Point2D object is valid, false otherwise.

int getNumberOfPoints();

Returns the total number of points stored in our vector.

RGPSegment2D getBoundingBox();

Returns Point2DImpl object with bounding box as a vector attribute.

iterator begin();

Returns an iterator defining the starting position of vector of points.

iterator end();

Returns an iterator defining the ending position of vector of points.

iterator:: (RGPPoint2D);*

Constructor of iterator, assigns the value of incoming pointer to the iterator calls variavle ptr.

RGPPoint2D iterator::operator();*

This is an overloading method for iterator, it returns the value of the iterator.

RGPPoint2D iterator::operator++(int);

This is also an overloading ++(post increment) of iterator, mainly used for moving the iterator to next element

RGPPoint2D iterator::operator++();

This is also an overloading ++(pre increment) of iterator, mainly used for moving the iterator to next element

bool iterator::operator!=(const iterator&);

This is an operator overloading for iterator, this checks if the given iterators are not equal.

bool iterator::operator==(const iterator&);

This is an operator overloading for iterator, this checks if the given iterators are not equal.

bool iterator::add(RGPPoint2D rgpp2d);

Returns true if our passed in RGP point is successfully added to our vector of points.

bool update(iterator it, RGPPoint2D rgpp2d);

Returns true if our passed in RGP point is successfully added to our vector of points using an iterator as index, false otherwise.

bool remove(iterator it);

Returns true if successfully removed RGP point using iterator as index from vector of point, false otherwise.

bool operator==(const Point2DImpl &p2d);

Returns true if two Point2Ds are equal, false otherwise.

bool operator!=(const Point2DImpl &p2d);

Returns true if two Point2Ds are not equal, false otherwise.

Point2DImpl operator[](int index);

Returns a specified RGP point at specified index.

Point2DImpl operator=(const Point2DImpl &p2dImpl);

Assigns the value of right Point2DImpl object to the left one

void pointSort(std::vector<RGPPoint2D> &bar);

Sorts a passed in vector of points into our stored vector of points.

void mergeSort(std::vector<RGPPoint2D> &left, std::vector<RGPPoint2D> &right, std::vector<RGPPoint2D> &bars);

Merges two passed in vector of points.

bool parseStringToVectorOfPoints(std::string st);

Returns true if the passed in string of points is successfully stored in vector of points, false otherwise.

Time Complexities:

- Most of the methods work in either linear or constant time.
- Methods isEmptyPoint(), getNumberofPoints(), all iterator functions and operator overloading takes constant time, i.e , O(1).
- Methods add(), remove(), printAllPoints(), getBoundingBox() take linear time, i.e, O(n) where n is the total number of coordinates.
- Methods like update(), parseStringToVectorOfPoints() take O(nlogn) time because we have to sort all the points, we are using merge sort.
- Therefore, creation of a point take O(nlogn) time and most of the operations can be done either in linear or constant time.

Kyler Ferrell-Clegg | Lokesh Surya Paluri | Sai Charan Kadari
CIS4930
Group 2

Line2DImpl

Convention to follow

(((a1,b1),(c1,d1)),((a2,b2),(c2,d2)),((a3,b3),(c3,d3)),((a4,b4),(c4,d4)))

Line2DImpl – Function Definitions

Line2DImpl();

The first constructor creates a Line2DImpl object but, only defines our pointer to our struct and doesn't store any segment information.

Line2DImpl(std::vector<RGPHalfSegment2D> listOfSegments);

The second constructor creates a Line2DImpl object based on a passed in vector of RGP half segments.

Line2DImpl(std::string listOfLine2DString);

The third constructor creates a Line2DImpl object based on a passed in string of segments.

Line2DImpl(std::ifstream& file);

The final constructor creates a Line2DImpl object based on a passed in file of segments.

~Line2DImpl();

The destructor clears any data that is stored in our vector of segments or our bounding box vector.

std::string getLineString();

Returns a string containing all of the line segments in our vector of segments.

void printAllLines();

Prints out every stored line in our vector of segments.

bool isEmptyLine();

Returns true if vector of segments is empty, false otherwise.

bool isValidLine();

Returns true if Line2D object is valid, false otherwise.

int getNumberOfSegments();

Returns the total number of segments stored in our vector.

RGPSegment2D getBoundingBox();

Returns Line2DImpl object with bounding box as a vector attribute.

iterator begin();

Returns an iterator defining the starting position of vector of segments.

iterator end();

Returns an iterator defining the ending position of vector of segments.

iterator::iterator(RGPHalfSegment2D);*

Assigns our RGPHalfSegment2D pointer in iterator class to incoming parameter.

RGPHalfSegment2D iterator::operator();*

Returns the value of the iterator.

RGPHalfSegment2D iterator::operator++(int);

This is also an overloading ++(post increment) of iterator, mainly used for moving the iterator to next element

RGPHalfSegment2D iterator::operator++();

This is also an overloading ++(pre increment) of iterator, mainly used for moving the iterator to next element

bool iterator::operator!=(const iterator&);

This is an operator overloading for iterator, this checks if the given iterators are not equal.

bool iterator::operator==(const iterator&);

This is an operator overloading for iterator, this checks if the given iterators are not equal.

bool add(RGPHalfSegment2D rgpSeg2d);

Returns true if our passed in RGP half segment is successfully added to our vector of segments.

bool update(iterator it, RGPHalfSegment2D rgpSeg2d);

Returns true if our passed in RGP half segment is successfully added to our vector of segments using an iterator as index, false otherwise.

bool remove(iterator it);

Returns true if successfully removed RGP half segment using iterator as index from vector of segments, false otherwise.

bool operator==(const Line2DImpl &l2d);

Returns true if two Line2Ds are equal, false otherwise.

bool operator!=(const Line2DImpl &l2d);

Returns true if two Line2Ds are not equal, false otherwise.

Line2DImpl operator[](int index);

Returns a specified RGP half segment at specified index.

Line2DImpl operator=(const Line2DImpl &l2dImpl);

Assigns the right Line2DImpl object to that in left

void lineSort(std::vector<RGPHalfSegment2D> &bar);

Sorts a passed in vector of segments into our stored vector of segments.

void mergeSort(std::vector<RGPHalfSegment2D> &left, std::vector<RGPHalfSegment2D> &right, std::vector<RGPHalfSegment2D> &bars);

Merges two passed in vector of segments.

bool parseStringToVectorOfLines(std::string st);

Returns true if the passed in string of lines is successfully stored in vector of segments, false otherwise.

Time Complexities:

- Most of the methods work in either linear or constant time.
- Methods isEmptyLine(), getNumberofSegments(), all iterator functions and operator overloading takes constant time, i.e , O(1).
- Methods remove(), printAllLines(), getBoundingBox() take linear time, i.e, O(n) where n is the total number of coordinates.
- Methods like add(), update(), parseStringToVectorOfLines() take O(nlogn) time because we have to sort all the halfsegments before we store them in our vector, we are using merge sort.
- Therefore, creation of a Line take O(nlogn) time and most of the operations can be done either in linear or constant time.

Kyler Ferrell-Clegg | Lokesh Surya Paluri | Sai Charan Kadari
CIS4930
Group 2

Region2DImpl

Convention:

[WCR((0 0,5 0,5 5,0 5,0 0),(1 1,3 1,3 3,1 3,1 1))&(2 2,2 3, 3 3,3 2,2 2)]

WCR is taken as a short form of 'Well Constructed Regions'

Note:

1. **Each bracket is a set of points which form the segments of a cycle.**
2. **First bracket contains the face and following brackets must represent the holes corresponding to that cycle**
3. **'&' represents a delimiter where the start of a new face and its cycle representation can be given**
4. **The input string is case sensitive and space-sensitive.+**
5. **Validity of region is tested by checking if the given region is edge-disjoint. Also, the areas of faces and holes are calculated and subtracted from one another to give a positive number result.**

Region2DImpl – Function Definitions

Region2DImpl();

The first constructor creates a Region2D object but, only defines our pointer to our struct and doesn't store any point information.

Region2D(std::string listOfRegion2DString);

The third constructor creates a Region2D object based on a passed in string of regions.

Throws exception when the region is interpreted as invalid or illegal.

~Region2D();

The destructor clears any data that is stored in our vector of vectors of annotated half segments or our bounding box vector.

Number area();

Returns a Number object representing area of our region object.

static bool isEmptyRegion(Region2D region);

Returns true if our passed in region is empty, false otherwise.

int getNumberOfFaces();

Returns the number of faces for or region object.

int getNumberOfHoles();

Returns the number of holes for or region object.

std::vector<RGPSegment2D> getBoundingBox();

Returns the bounding box of our region object.

bool operator==(const Region2D &p2d);

Returns true if passed in region object is equal to region object, false otherwise.

bool operator!=(const Region2D &p2d);

Returns true if passed in region object is not equal to region object, false otherwise.

bool addFace(std::vector<RGPSegment2D>);

Adds a face to a region object.

Throws exception when the face renders the region object as invalid

bool parseWDR(std::string inputString);

Parses string to get the complete region object.

Number getAreaOfCycle(std::vector<RGPAnnotatedHalfSegment2D> vectorOfSegments);

Returns the area of a passed in cycle.

std::vector<RGPAnnotatedHalfSegment2D> constructRegion(std::string formattedInputString);

Returns a constructed region based on passed in string.

bool doSegmentsIntersect(RGPSegment2D a, RGPSegment2D b);

Checks if both RGPSegments intersect

bool onSegment(RGPPoint2D pi, RGPPoint2D pj, RGPPoint2D pk);

Checks if the point pk is on segment with points pi and pj.

void sortAnnotatedHS(std::vector<RGPAnnotatedHalfSegment2D> vec);

Sorts the given vector of Annotated half segments.

Kyler Ferrell-Clegg | Lokesh Surya Paluri | Sai Charan Kadari
CIS4930
Group 2

Additionally iterators are present for holes and faces separately to get all faces and holes as required.

Time Complexities:

- Methods like getNumberofFaces(), getNumberofHoles(), isEmptyRegion() and all operator overloading work in constant time.
- All the other methods are related to the construction in a direct or indirect way, so all of them have a complexity of O(n^2). It has this complexity because the intersection between cycles has been done in a way that we check if any of the whole set of segments intersect
- Therefore, creation takes O(n^2) time and all other functions take linear time.

Point2DForProgrammer and Point2D – Function Definitions

- All the functions in these 4 files, namely Point2DForProgrammer.h, Point2DForUser.h, Point2DForProgrammer.cpp and Point2DForUser.cpp, are directly taken from the Point2DImpl file. All the definition and function remains the same.
- In these files we create an instance of Point2Dimpl and call its functions based on invocations.

Line2DForProgrammer and Line2D– Function Definitions

- All the functions in these 4 files , namely Line2DForProgrammer.h, Line2DForUser.h, Line2DForProgrammer.cpp and Line2DForUser.cpp, are directly taken from the LineImpl file. All the definition and function remains the same.
- In these files we create an instance of Line2Dimpl object and call its functions based on invocations in these headers.

Region2DForProgrammer and Region2D – Function Definitions

- All the functions in these both files, , namely Region2DForProgrammer.h, Region2DForUser.h, Region2DForProgrammer.cpp and Region2DForUser.cpp, are directly taken from the Impl file. All the definition and function remains the same.
- In these files we create an instance of Region2Dimpl object and call its functions based on invocations in these headers.

Kyler Ferrell-Clegg | Lokesh Surya Paluri | Sai Charan Kadari
CIS4930
Group 2

Testing Phase:

- Testing has been done on all the implemented code, with major edge case and results.
- Performance of the code has been given above.
- We have also included 3 main() files which cover the complete functionality of the code.

Main for Point:

```
int main(void)
{
        //adding through strings.
        std::string str = "((333.33,22.22),(12,6),(3,4),(9,7),(4,5))";
        Point2DImpl x(str);
        std::string str2 = "((9,7),(333.33,22.22),(3,4),(4,5),(12,6))";
        Point2DImpl y(str2);
```

```
	//adding through strings when input format is not correct. Checked all other cases as well
	std::string str3 = "((333.3322.22),(4,5),(3,4),(4,5))";
	Point2DImpl z(str3);

	//method to print all the points in our object

	std::cout<<"All points in x are"<<std::endl;
	x.printAllPoints();
	std::cout<<std::endl;

	//checking operator !=
	std::cout<<"check if x!=y"<<std::endl;
	if(x!=y)
		std::cout<<"true"<<std::endl;
	else
		std::cout<<"false"<<std::endl;

	//checking operator ==
	std::cout<<"check if x==y"<<std::endl;
	if(x==y)
		std::cout<<"true"<<std::endl;
	else
		std::cout<<"false"<<std::endl;

	//check if the Point2D object is Empty
```

```
	bool empty = x.isEmptyPoint();
	std::cout<<"output after checking if empty is : "<<empty<<std::endl;

	//get the total number of points in the object
	std::cout<<"total number of points in x are : "<<x.getNumberOfPoints()<<std::endl;

	//working of Point2DImpl iterator. This covers all methods of iterator like end(), begin(),
!=, ==, *
	std::cout<<"working of iterator and printing all the points in the vector"<<std::endl;
	for(Point2DImpl::iterator it = x.begin(); it!= x.end();++it)
		std::cout<<*it;
	std::cout<<std::endl;

	//creation of point and adding it to the Point2DImpl object
	RGPPoint2D pt(Number("31"),Number("22"));
	x.add(pt);
	//output after adding
	std::cout<<"object after adding point"<<std::endl;
	for(Point2DImpl::iterator it = x.begin(); it!= x.end();++it)
		std::cout<<*it;
	std::cout<<std::endl;

	//working of remove method for Point2DImpl object
	Point2DImpl::iterator it = x.begin();
	it++;
	it++;
	x.remove(it);
```

```
	std::cout<<"object after removing point"<<std::endl;
	for(Point2DImpl::iterator it = x.begin(); it!= x.end();++it)
		std::cout<<*it;
	std::cout<<std::endl;

	//working of update method for Point2DImpl
	Point2DImpl::iterator it1 = x.begin();
	it1++;
	RGPPoint2D pt1(Number("23"),Number("2"));
	x.update(it1,pt1);
	std::cout<<"object after updating point"<<std::endl;
	for(Point2DImpl::iterator it = x.begin(); it!= x.end();++it)
		std::cout<<*it;
	std::cout<<std::endl;

	//working of = operator for Point2DImpl
	x=y;
	//x was updated multiple times, now it is equal to y
	std::cout<<"output after doing ="<<std::endl;
	for(Point2DImpl::iterator it = x.begin(); it!= x.end();++it)
		std::cout<<*it;
	std::cout<<std::endl;

	//working of [] in point2DImpl
	Point2DImpl t;
	t = y[2];
```

```
        Point2DImpl::iterator it2 = t.begin();
        std::cout<<"index works and the point added was "<<*it2<<std::endl;
        //code for getting the bounding box of the set of points
        Line2DImpl bx;
        bx = y.getBoundingBox();
        std::cout<<"segment in the bounding box"<<std::endl;
        for(Line2DImpl::iterator it = bx.begin(); it!= bx.end();it++)
                std::cout<<(*it).segment;
        std::cout<<std::endl;
        return 0;
}
```

Output:

All points are added successfully

All points are added successfully

Input Format is not correct

All points in x are

((3,4)(4,5)(9,7)(12,6)(333.33,22.22))

check if x!=y

false

check if x==y

true

output after checking if empty is : 0

total number of points in x are : 5

working of iterator and printing all the points in the vector

(3,4)(4,5)(9,7)(12,6)

object after adding point

Kyler Ferrell-Clegg | Lokesh Surya Paluri | Sai Charan Kadari
CIS4930
Group 2

(3,4)(4,5)(9,7)(12,6)(31,22)

object after removing point

(3,4)(4,5)(12,6)(31,22)

object after updating point

(3,4)(12,6)(23,2)(31,22)

output after doing =

(3,4)(4,5)(9,7)(12,6)

index works and the point added was (9,7)

segment in the bounding box

(3,4),(333.33,22.22)

Main for Line:

```
int main()
{
	//adding through strings.
	std::string str = "(((3,4),(2,2)),((6,2),(5,3)),((6,1),(7,9)),((8,4),(6,6)))";
	Line2DImpl ln(str);
	std::string str2 = "(((3,4),(2,2)),((6,2),(5,3)),((6,1),(7,9)),((8,4),(6,6)))";
	Line2DImpl ln2(str2);

	//adding through strings when input format is not correct
	std::string str3 = "(((3,4),(2,2)),((6,2),(5,3)),((6,1)(7,9)),((8,4),(6,6)))";
	Line2DImpl wrongInp(str3);

	//checking operator !=
	std::cout<<"check if ln1!=ln2"<<std::endl;
```

```
	if(ln!=ln2)
		std::cout<<"true"<<std::endl;
	else
		std::cout<<"false"<<std::endl;

	//checking operator ==
	std::cout<<"check if ln1==ln2"<<std::endl;
	if(ln==ln2)
		std::cout<<"true"<<std::endl;
	else
		std::cout<<"false"<<std::endl;

	//checking if any object is empty
	bool empty = ln.isEmptyLine();
	std::cout<<"check if ln is empty : "<<empty<<std::endl;

	//method to get total number of segments in the object
	std::cout<<"total number of segments in our object are :
"<<ln.getNumberOfSegments()<<empty<<std::endl;

	//checking the functionality of an iterator
	std::cout<<"Travesed segments through iterators"<<std::endl;
	for(Line2DImpl::iterator it = ln.begin(); it!=ln.end(); it++)
	{
		std::cout<<(*it).segment;
	}
	std::cout<<std::endl;
```

```
//adding a segment to line2D
RGPPoint2D p1(Number("21"),Number("31"));
RGPPoint2D p2(Number("22"),Number("33"));
RGPSegment2D seg(p1,p2);
std::cout<<"list after adding"<<std::endl;
ln.add(seg);
for(Line2DImpl::iterator it = ln.begin(); it!=ln.end(); it++)
{
        std::cout<<(*it).segment;
}
std::cout<<std::endl;

//code for removing a segment from an object
Line2DImpl::iterator it = ln.begin();
it++;
it++;
ln.remove(it);
std::cout<<"list after removing the segment"<<std::endl;
for(Line2DImpl::iterator it = ln.begin(); it!=ln.end(); it++)
{
        std::cout<<(*it).segment;
}
std::cout<<std::endl;

//code for updating a segment through iterator
```

```
Line2DImpl::iterator it1 = ln.begin();
it1++;
RGPPoint2D p3(Number("12"),Number("5"));
RGPPoint2D p4(Number("2"),Number("5"));
RGPSegment2D seg2(p3,p4);
std::cout<<"list of segments after updating "<<std::endl;
ln.update(it1,seg2);
for(auto it = ln.begin(); it!=ln.end(); it++)
{
        std::cout<<(*it).segment;
}
std::cout<<std::endl;

//checking functionality of index and "=" operator overloading
Line2DImpl ln3;
ln3 = ln[2];
Line2DImpl::iterator it4 = ln3.begin();
std::cout<<"after assigning the vector is "<<std::endl;
std::cout<<(*it4).segment<<std::endl;

//checking method to print all lines
std::cout<<"method to print all the segments in our object"<<std::endl;
ln.printAllLines();
std::cout<<std::endl;

//method to get the bounding boxe
```

```
        Line2DImpl ln4 = ln.getBoundingBox();
        Line2DImpl::iterator it5 = ln4.begin();
        std::cout<<"bounding box is "<<std::endl;
        std::cout<<(*it5).segment<<std::endl;
        return 0;
}
```

Output:

success

success

failed

check if ln1!=ln2

false

check if ln1==ln2

true

check if ln is empty : 0

total number of segments in our object are : 40

Travesed segments through iterators

(2,2),(3,4)(2,2),(3,4)(5,3),(6,2)(6,1),(7,9)(5,3),(6,2)(6,6),(8,4)(6,1),(7,9)

list after adding

(2,2),(3,4)(2,2),(3,4)(5,3),(6,2)(6,1),(7,9)(5,3),(6,2)(6,6),(8,4)(6,1),(7,9)(6,6),(8,4)(21,31),(22,33)

list after removing the segment

(2,2),(3,4)(2,2),(3,4)(5,3),(6,2)(6,6),(8,4)(6,1),(7,9)(6,6),(8,4)(21,31),(22,33)

list of segments after updating

(2,2),(3,4)(2,5),(12,5)(6,1),(7,9)(6,6),(8,4)(2,5),(12,5)(21,31),(22,33)

after assigning the vector is

(6,1),(7,9)

method to print all the segments in our object

((2,2),(3,4)(2,5),(12,5)(6,1),(7,9)(6,6),(8,4)(2,5),(12,5)(21,31),(22,33)(21,31),(22,33))

bounding box is

(2,2),(22,33)

Main for Region:

```
Int main()
{
        std::string regStr = "[WCR((0 0,5 0,5 5,0 5,0 0),(1 1,3 1,3 3,1 3,1 1))]";
        std::cout<<"region to be added: "<<regStr<<std::endl;
        Region2DImpl r(regStr);
        std::cout<<"area of the region: "<< r.area() <<std::endl;
        std::cout<<"number of faces in region: "<< r.getNumberOfFaces() <<std::endl;
        std::cout<<"number of holes in region: "<< r.getNumberOfHoles() <<std::endl;
        std::cout<<"is the region empty? "<< r.isEmptyRegion() <<std::endl;

        Region2DImpl::iteratorforFaces it3 = r.beginFaces();
        for(Region2DImpl::iteratorforFaces it = r.beginFaces(); it!= r.endFaces();it++)
        {
                std::vector<RGPAnnotatedHalfSegment2D> vec = *it;
```

```
            std::cout<<vec[0].segment;
        }
        std::cout<<"iterator works over the faces. Printing the points on first segment:
"<<(*it3)[0].segment<<std::endl;


        Region2DImpl::iteratorforHoles it4 = r.beginHoles();
        for(Region2DImpl::iteratorforHoles it = r.beginHoles(); it!= r.endHoles();it++)
        {
            std::vector<RGPAnnotatedHalfSegment2D> vec = *it;
            std::cout<<vec[0].segment;
        }
        std::cout<<"iterator works over the Holes. Printing the points on first segment:
"<<(*it4)[0].segment<<std::endl;
        std::cout<<"diagonal of the bounding box for the region:
"<<r.getBoundingBox()<<std::endl;


        std::string invalidRegString = "[WCR((0 0,1 1,2 2))]";
        try
        {
            std::cout<< "Trying to construct an invalid/illegal region: " << std::endl;
            Region2DImpl reg(invalidRegString);
        }
        catch (const std::exception &exc)
        {
            std::cerr << "Exception caught while creating region: " <<exc.what() << std::endl;
        }
```

```
        std::cout << "All successfull!" << std::endl;

        return 0;

}
```

Output:

region to be added: [WCR((0 0,5 0,5 5,0 5,0 0),(1 1,3 1,3 3,1 3,1 1))]

area of the region: 21

number of faces in region: 1

number of holes in region: 1

is the region empty? 0

iterator works over the faces. Printing the points on first segment: (0,0),(5,0)

iterator works over the Holes. Printing the points on first segment: (1,1),(3,1)

diagonal of the bounding box for the region: (0,0),(1,5)

Trying to construct an invalid/illegal region:

Exception caught while creating region: Invalid Constructed Region

All successfull!