Algorithmics and C Programming

LO27 – Project

2D Polygon Library

Written Report

TSAGALOS Thibault
DURANCE Aurélien

A2014

Table des matières

1. Aim of the Project 3
2. Used Data Structures 4
3. Functions 7
A point on the Fonctions : 16
Conclusion 17

1. Aim of the Project

The aim of this project was to create a library in c, in order to create and manipulate 2D-polygons (union, rotation, etc…). A polygon is an object made of points (at least 3 for a complete polygon) and those points are characterized by their abscissa x, and their ordinate y.

2. Used Data Structures

To manage the creation and the manipulation of polygons, we created several data structures which are listed below.

- <u>The Point data structure :</u>

This data structure corresponds to a mathematical point. A point is characterized by its two coordinates, x and y, which are double variables in C code.

Here you can see our declaration of this data structure in C code:

```
typedef struct{
        double  x;
        double  y;
} Point;
```

- <u>The Element data structure :</u>

This data structure is an Element of a polygon, composed of a point, a pointer pointing on another Element, which is the next point in the polygon, and another pointer for the previous element.

Here you can see our declaration of this data structure in C code:

```
typedef struct elem{
        Point value;
        struct elem* next;
        struct elem* prev;
} Element;
```

- <u>The Polygon data structure:</u>

This data structure is the representation of a polygon, composed of a pointer on an Element which is the first point of the polygon and of an integer variable called “length” which is the actual length of the polygon which can be very useful in some function where you need to access to the last point of a given polygon.

Here you can see our declaration of this data structure in C code:

```
typedef struct {
   Element* head;
   int length;
} Polygon;
```

- <u>The Boolean data structure </u>:

No need to introduce you this one, the well-known Boolean! You can find it in the header file named “boolean.h”.

- The Status data structure :

This data structure is used to compare the location of two polygons especially in the function “containsPolygon”. You can find this data structure in the header file named “status.h”

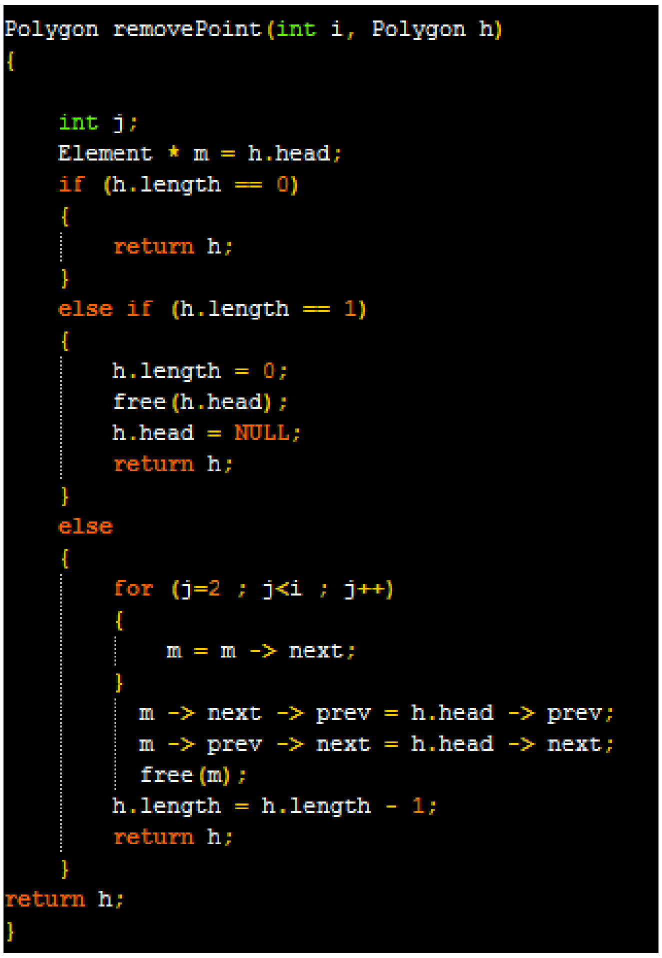
Here you can see our declaration of this data structure in C code:

```
typedef enum {

   INSIDE = 0,

   OUTSIDE = 1,

   INTERSECT = 2,

   ENCLOSING = 3,

   EQUAL = 4,

} Status;
```

3. Functions

Now we will present you some of the functions provided in our 2D-polygon library. We will explain what these functions do and what the difficulties we encountered were.

First, we'll present you the fonction removePoint.



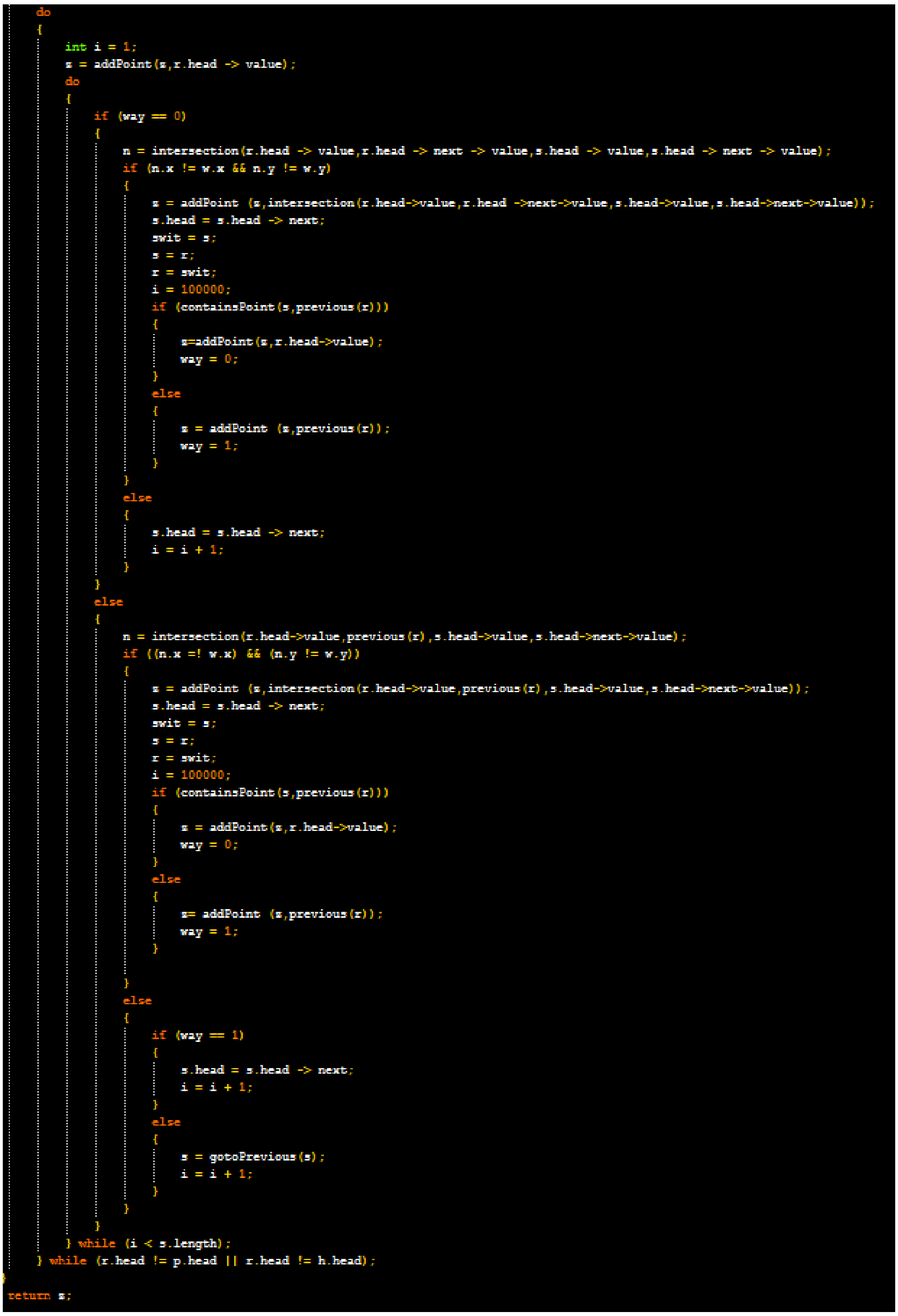
```
Polygon removePoint(int i, Polygon h)
{

    int j;
    Element * m = h.head;
    if (h.length == 0)
    {
        return h;
    }
    else if (h.length == 1)
    {
        h.length = 0;
        free(h.head);
        h.head = NULL;
        return h;
    }
    else
    {
        for (j=2 ; j<i ; j++)
        {
            m = m -> next;
        }
          m -> next -> prev = h.head -> prev;
          m -> prev -> next = h.head -> next;
          free(m);
        h.length = h.length - 1;
        return h;
    }
return h;
}
```

Fonction removeP oint : Polygon x Integer -> Polygon, removes the ith pointfrom the specified polygon (points are indexed from 1 to N in a polygon) .

This fonction just go through the polygon until it reach the i'th point, then suppress this point and ajusting the length.

We didn't encountered much difficulties for this one.

Fonction UnionPolygon : Polygon x Polygon -> Polygon, Computes the union between two specified polygons.



```
    do
    {
        int i = 1;
        z = addPoint(z,r.head -> value);
        do
        {
            if (way == 0)
            {
                n = intersection(r.head -> value,r.head -> next -> value,s.head -> value,s.head -> next -> value);
                if (n.x != w.x && n.y != w.y)
                {
                    z = addPoint (z,intersection(r.head->value,r.head ->next->value,s.head->value,s.head->next->value));
                    s.head = s.head -> next;
                    swit = s;
                    s = r;
                    r = swit;
                    i = 100000;
                    if (containsPoint(s,previous(r)))
                    {
                        z=addPoint(z,r.head->value);
                        way = 0;
                    }
                    else
                    {
                        z = addPoint (z,previous(r));
                        way = 1;
                    }
                }
                else
                {
                    s.head = s.head -> next;
                    i = i + 1;
                }
            }
            else
            {
                n = intersection(r.head->value,previous(r),s.head->value,s.head->next->value);
                if ((n.x =! w.x) && (n.y != w.y))
                {
                    z = addPoint (z,intersection(r.head->value,previous(r),s.head->value,s.head->next->value));
                    s.head = s.head -> next;
                    swit = s;
                    s = r;
                    r = swit;
                    i = 100000;
                    if (containsPoint(s,previous(r)))
                    {
                        z = addPoint(z,r.head->value);
                        way = 0;
                    }
                    else
                    {
                        z= addPoint (z,previous(r));
                        way = 1;
                    }

                }
                else
                {
                    if (way == 1)
                    {
                        s.head = s.head -> next;
                        i = i + 1;
                    }
                    else
                    {
                        s = gotoPrevious(s);
                        i = i + 1;
                    }
                }
            }
        } while (i < s.length);
    } while (r.head != p.head || r.head != h.head);
}
return z;
```

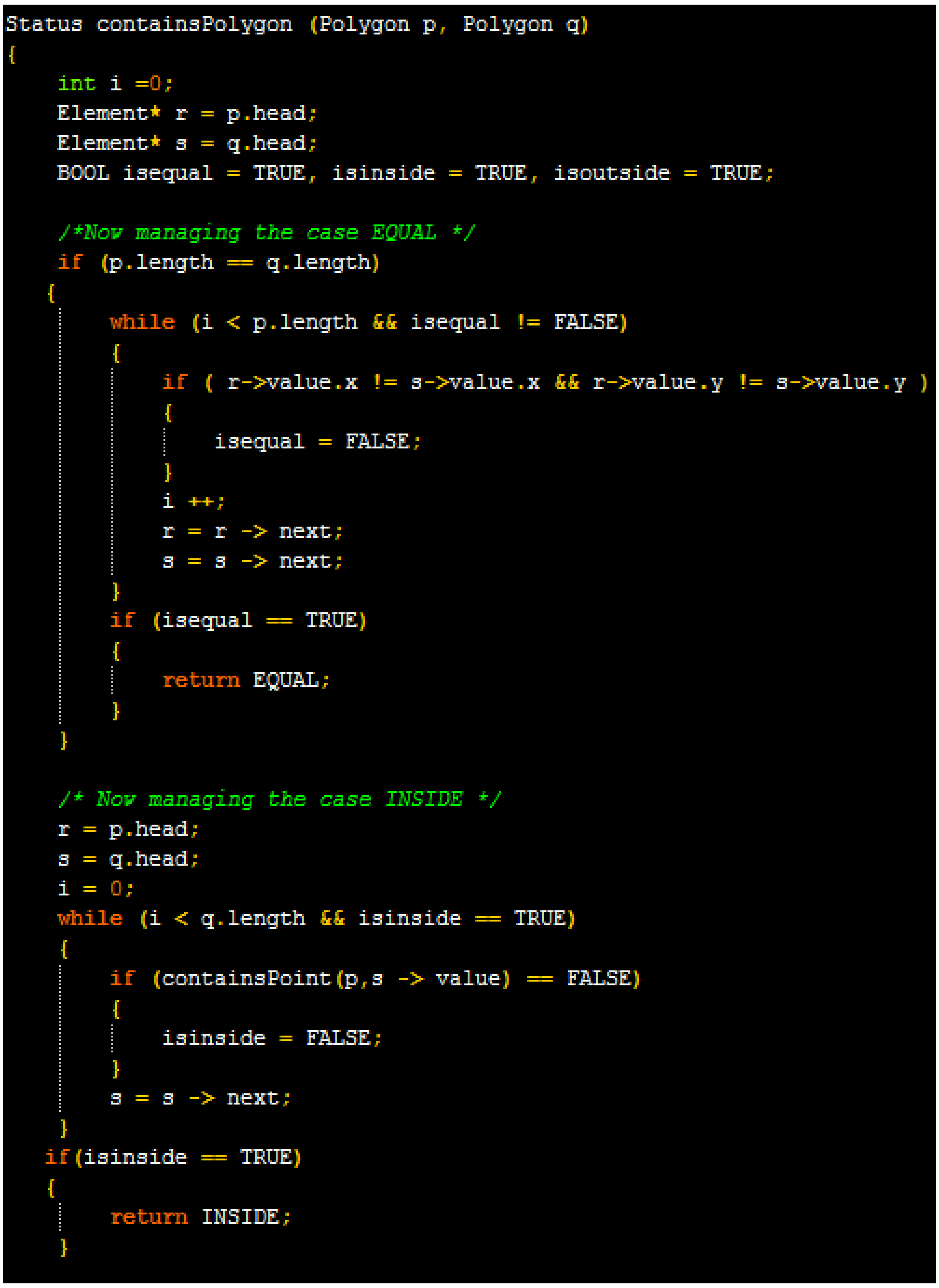
In this function, we first manage a few trivial cases : if the polygons are inside one in another, if they are completely separed, etc, thanks to the function containsPolygon that we will explain later.

Then, we made two loops (on the screenshot) :
Those loops are comparing, each segment of the first polygon (one by one) to each segments of the second polygon.
If the segments don't cut, we add the points of the segment of the first polygon in the result polygon.
If the segments are cutting each other, we add the intersection to the final polygon, and then we need to go in the direction of the point which isn't in the polygon. We add this point and start again the process, but inversing both polygons (the new first polygon will start just after the intersection).
We just continue until we reach the beginning of one of the polygons.

We encountered a lot of difficulties for this function, which was, for us, the most difficult to do.
First, we had a lot of troubles to find a mathematical way to answer to that problem (more than 10 hours, everything that we tried failed), and then we had as much problems to implement it in c, trying to make the program do what we want was very hard.

For the function intersection, it's the same thing but we need to go always inside the polygon, until we reach the first intersection.

Fonction containsPolygon : Polygon x Polygon -> Status, return a Status enumeration type that could take several values.



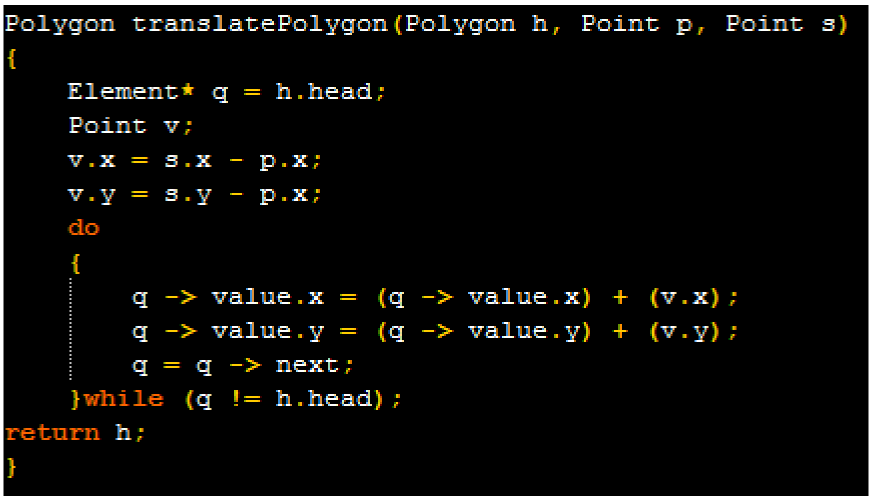
```
Status containsPolygon (Polygon p, Polygon q)
{
    int i =0;
    Element* r = p.head;
    Element* s = q.head;
    BOOL isequal = TRUE, isinside = TRUE, isoutside = TRUE;

    /*Now managing the case EQUAL */
    if (p.length == q.length)
   {
        while (i < p.length && isequal != FALSE)
        {
            if ( r->value.x != s->value.x && r->value.y != s->value.y )
            {
                isequal = FALSE;
            }
            i ++;
            r = r -> next;
            s = s -> next;
        }
        if (isequal == TRUE)
        {
            return EQUAL;
        }
    }

    /* Now managing the case INSIDE */
    r = p.head;
    s = q.head;
    i = 0;
    while (i < q.length && isinside == TRUE)
    {
        if (containsPoint(p,s -> value) == FALSE)
        {
            isinside = FALSE;
        }
        s = s -> next;
    }
   if(isinside == TRUE)
   {
        return INSIDE;
    }
```

We test here, one by one, each case.

First, the case EQUAL : if the size, and each point are equal, the polygons are equal.
Then, the case INSIDE : with a fonction isinside, we check if each point of a polygon is inside an other one. If it’s the case, we return inside.
Those two cases are illustrated on ther screenshot above.
After, the case OUTSIDE : if each point of a polygon isn’t inside another polygon, we return outside.
Then, the same thing than INSIDE, but reversing the two polygons says if it is the cas ENCLOSING.
Finally, if it is none of the case above, we conclude by saying it’s the case INTERSECT.

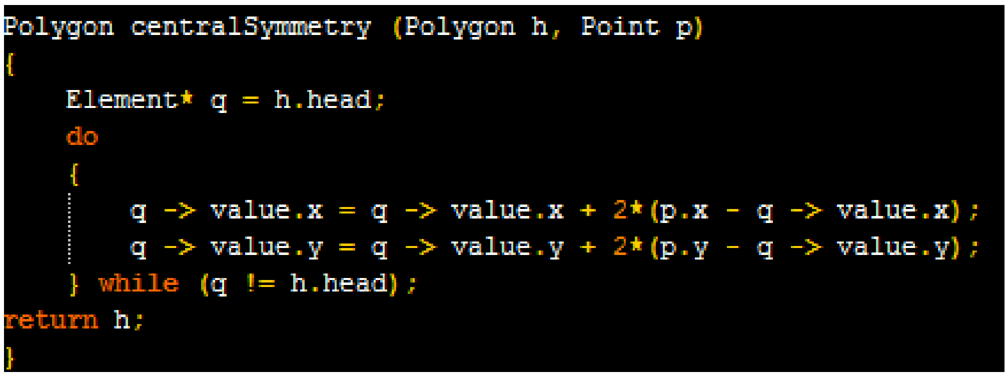
We encountered here some difficulties, because they were a lot of cases to think.

Fonction TranslatePolygon : Polygon x Point x Point –> Polygon, computes the translation of the specified polygon according to the vector defined by the two specified points.


```
Polygon translatePolygon(Polygon h, Point p, Point s)
{
    Element* q = h.head;
    Point v;
    v.x = s.x - p.x;
    v.y = s.y - p.x;
    do
    {
        q -> value.x = (q -> value.x) + (v.x);
        q -> value.y = (q -> value.y) + (v.y);
        q = q -> next;
    }while (q != h.head);
return h;
}
```

This fonction take each point of a polygon, and applies the vector defined by te two specified points to it.
Once we found the mathematical solution to this one, the implementation in c wasn't that hard.

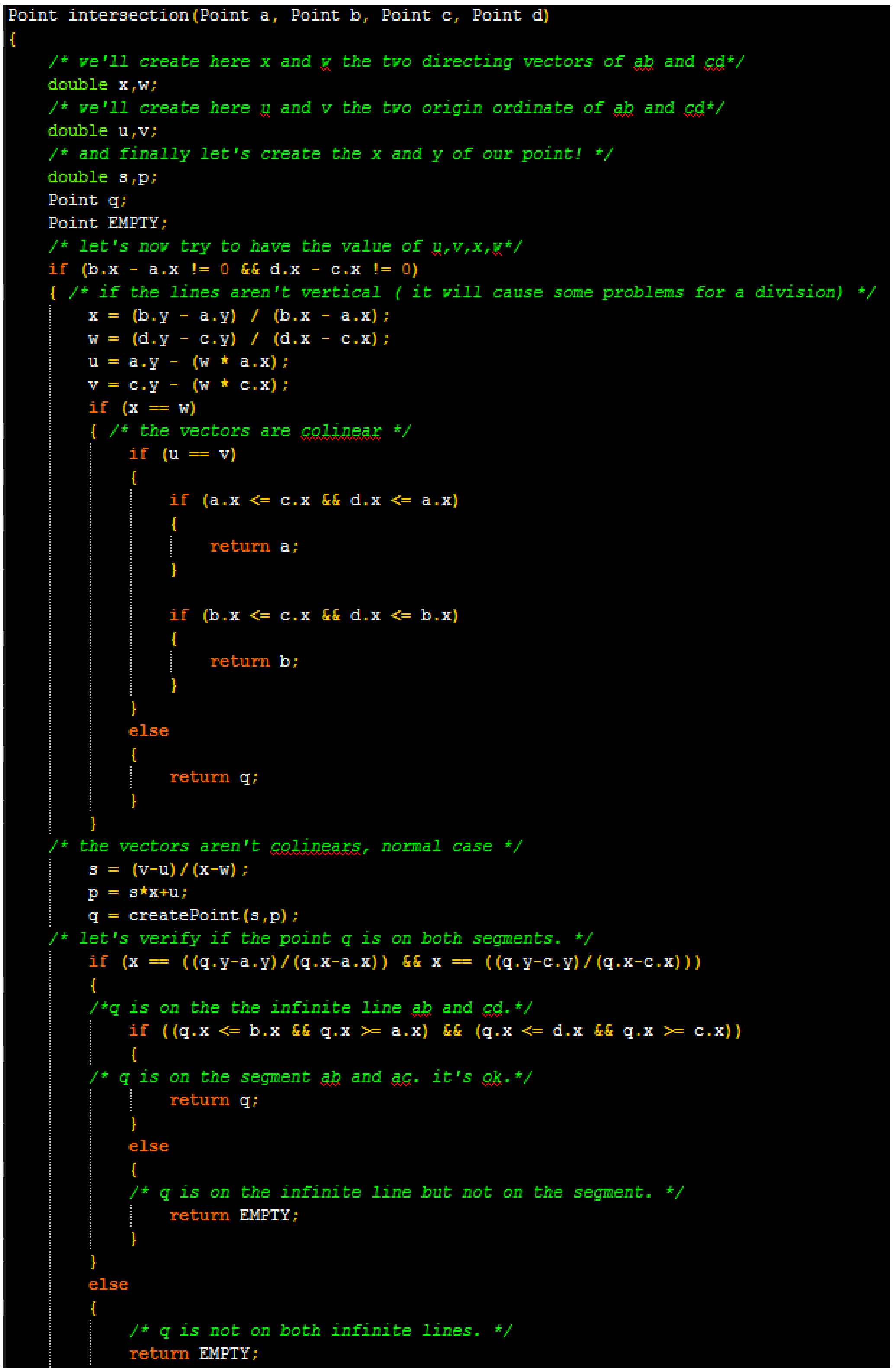
Fonction centralSymmetry : Polygon x Point -> Polygon, computes the central symmetry of the specified polygon according to the specified point.



```
Polygon centralSymmetry (Polygon h, Point p)
{
    Element* q = h.head;
    do
    {
        q -> value.x = q -> value.x + 2*(p.x - q -> value.x);
        q -> value.y = q -> value.y + 2*(p.y - q -> value.y);
    } while (q != h.head);
return h;
}
```

We just take each point, and apply two times the vector between the point and the point given for the central symmetry.
Once again, simple to implement once we had the mathematical solution.

Fonction Intersection : Point x Point x Point x Point -> Point. This fonction takes two segments, caracterized by their extremities (points), and gives back a poinr with nothing inside if there is no intersection, or the intersection between the two segments.



```
Point intersection(Point a, Point b, Point c, Point d)
{
    /* we'll create here x and y the two directing vectors of ab and cd*/
    double x,w;
    /* we'll create here u and v the two origin ordinate of ab and cd*/
    double u,v;
    /* and finally let's create the x and y of our point! */
    double s,p;
    Point q;
    Point EMPTY;
    /* let's now try to have the value of u,v,x,y*/
    if (b.x - a.x != 0 && d.x - c.x != 0)
    { /* if the lines aren't vertical ( it will cause some problems for a division) */
        x = (b.y - a.y) / (b.x - a.x);
        w = (d.y - c.y) / (d.x - c.x);
        u = a.y - (w * a.x);
        v = c.y - (w * c.x);
        if (x == w)
        { /* the vectors are colinear */
            if (u == v)
            {
                if (a.x <= c.x && d.x <= a.x)
                {
                    return a;
                }

                if (b.x <= c.x && d.x <= b.x)
                {
                    return b;
                }
            }
            else
            {
                return q;
            }
        }
    /* the vectors aren't colinears, normal case */
        s = (v-u)/(x-w);
        p = s*x+u;
        q = createPoint(s,p);
    /* let's verify if the point q is on both segments. */
        if (x == ((q.y-a.y)/(q.x-a.x)) && x == ((q.y-c.y)/(q.x-c.x)))
        {
        /*q is on the the infinite line ab and cd.*/
            if ((q.x <= b.x && q.x >= a.x) && (q.x <= d.x && q.x >= c.x))
            {
        /* q is on the segment ab and ac. it's ok.*/
                return q;
            }
            else
            {
            /* q is on the infinite line but not on the segment. */
                return EMPTY;
            }
        }
        else
        {
            /* q is not on both infinite lines. */
            return EMPTY;
```

For this fonction, we had to be carefull for many cases : we devide a lot, we need to exclude the 0 division, or find an alternate solution for it, and we need to manage some trivial cases.

First, if the lines aren't vertical, (to avoid a division by 0), we find the two directing vectors and origin coordinates for the two 'segments', like if they were lines. (on the screenshot)

If the vectors are colinear,we need to check if the lines cut themselves, and if they do, on which extremity : we return the extremity concerned.

Then, if they aren't colinear, we use mathematicals fomulas to find the hypothetical intersection point, and then verify that this point is on the two segments. If it's the case, we return this point. (also on the screenshot)

Finally, if the lines are verticals, it's the same thing than if the lines are colinear.

<u>Note</u> : The fonctions scalePolygon, rotationPolygon, centralSymmetry, and translatePolygon erase the initial polygon to input the final one, trying to optimize the utilisation of memory.

A point on the Fonctions :

The fonctions createPoint, removePoint, createPolygon, addPoint, centralSymmetry, rotatePolygon, containsPoint, rotatePolygon, translatePolygon, printPoint, printPolygon are working well.

The fonction containsPolygon is working, but sometimes gives wrong answers, or bug the system.
The fonction toString seems to work, but we had a memory problem so we couldn’t test it.

The fonctions unionPolygons, intersectionPolygons, differencePolygons and exclusiveORPolygons aren’t working. But we let the code in, to let you see the work that we did.

Conclusion

This project was very interesting, we could see the difficulties encountered by developpers, and to teach us a lot about algorithms and C language.

We had a lot of difficulties making fonctionnal fonctions, the fonction is working (mathematically), but then had some bugs when we launched it in c.

Teamworking was also very interesting, we could have a different view on all our work, what is very usefull.

Report by DURANCE Aurélien and TSAGALOS Thibault