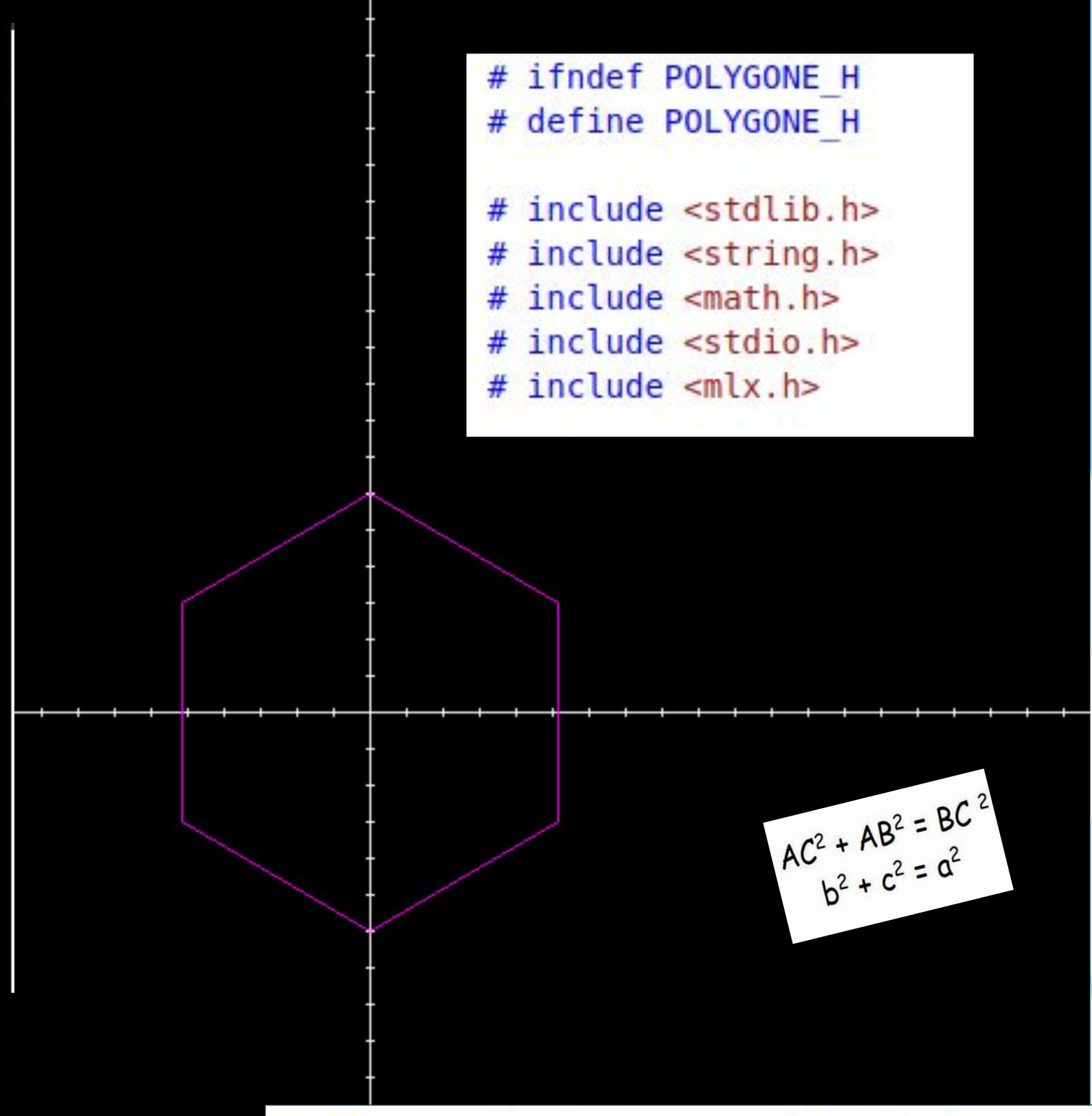
Projet D'Algorithmique avancé :

POLYGONE

```
*****Les Points:*****
Size: 6
Type:   XX

A (x.0.00; y.6.00)
B (x.5.20; y.3.00)
C (x.5.20; y.-3.00)
D (x.0.00; y.-6.00)
E (x.-5.20; y.-3.00)
F (x.-5.20; y.3.00)

*****Les Segements(cotes):*****
AB = 6.00
BC = 6.00
CD = 6.00
DE = 6.00
EF = 6.00
FA = 6.00

*****Les Angles:*****
 B = 119.98
 C = 119.98
 D = 120.04
 E = 119.98
 F = 119.98
 A = 120.04

*****Les Caracteristiques::*****

Perimetre:      36.01
Surface:        93.60
Regulier?       1

******************************
```



```
# ifndef POLYGONE_H
# define POLYGONE_H

# include <stdlib.h>
# include <string.h>
# include <math.h>
# include <stdio.h>
# include <mlx.h>
```

$$AC^2 + AB^2 = BC^2$$
$$b^2 + c^2 = a^2$$

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
float   kachi(t_point A, t_point B, t_point C)
{
    float   AB, AC, BC;
    float   k;

    AB = powf((B.x - A.x), 2)+ powf((B.y - A.y), 2);
    AC = powf((C.x - A.x), 2)+ powf((C.y - A.y), 2);
    BC = powf((C.x - B.x), 2)+ powf((C.y - B.y), 2);

    k = (AB + BC - AC) / (2 * sqrtf(AB) * sqrtf(BC));
    return (k);
}
```


Réalisé par :

- *EL KHALILI Ibrahim (N° 24)*
- *EL MAHAOUI Abdel HAdi* (N°26)

Encadré par :

- *Pr. DARGHAM Abdelmajid*

Remerciement :

Au terme de ce Projet je tient à présenter mes vifs remerciements à tous ceux qui ont contribué de prés ou de loin à sa réalisation.

On m’adresse surtout à notre encadrant, Pr. DARGHAM et on tient à lui exprimer notre profonde gratitude et notre reconnaissance pour l’aide qu’ils nous ont apportée tout au long de ce travail.

Ce document a pour but de décrire le déroulement de notre projet d’informatique en langage C permettant la création d’un polygone.

Sommaire :

1-Introduction :

Un polygone est une figure géométrique fermée composée de plusieurs segments. On se restreint aux polygones convexes. Réaliser un module pour le traitement des polygones convexes en utilisant les listes chaînées. Le module doit fournir les opérations suivantes : créer un polygone, calculer le périmètre et la surface d'un polygone, tester si un polygone est un triangle et le qualifier (isocèle, équilatéral, rectangle, isocèle rectangle), tester si un polygone est un quadrilatère et le qualifier (trapèze, parallélogramme, carré, rectangle, losange), tester si un point est à l'extérieur, à l'intérieur ou sur un côté d'un polygone, tester si deux polygones sont similaires (leurs côtés correspondants sont proportionnels et leurs angles correspondants sont égaux), tester si un polygone est régulier (ses côtés sont isométriques et ses angles sont égaux), tester si un polygone est à l'intérieur d'un autre.

2- Présentation des algorithmes :

Notre programme passe par plusieurs étapes avant de dessiner notre polygone et afficher ses paramètres:

- ***Stockage ds points a partir d'un fichier txt;***

Au premier lieu on convertit le fichier en une liste chaînée avec laquelle on travaille dans notre programme.

- ***Verification des points;***

On s'assure que dans notre liste il n'existe pas des points alignés; en calculant les angles du polygone et éliminer les points avec l'angle 180º.

- ***Calcul des différentes caractéristiques du polygone;***

Nombres des points (size): la longueur de la liste chaînée est le nombre des sommets du polygone.

Segments : on calcule la distance entre les points a l'aide des coordonnées.

Angles : à l'aide du théorème d'AL KASHI, on peut trouver le cosinus d'un angle à partir des distance entre les trois points créant cet angle.

Surface & Périmètre: le calcul du périmètre est facile après que nous avons déjà la longueure des segments. La surface est calculée à l'aide des coordonnées des différents points.

Régularité du polygone: la vérification de la régularité passe par deux étapes, premièrement tous les segments doivent avoir la même distance, si cette condition est vérifiée on passe à la vérification d'égalité des angles, sinon notre polygone n'est pas régulier.

- ***Traitement des Triangles (size = 3);***

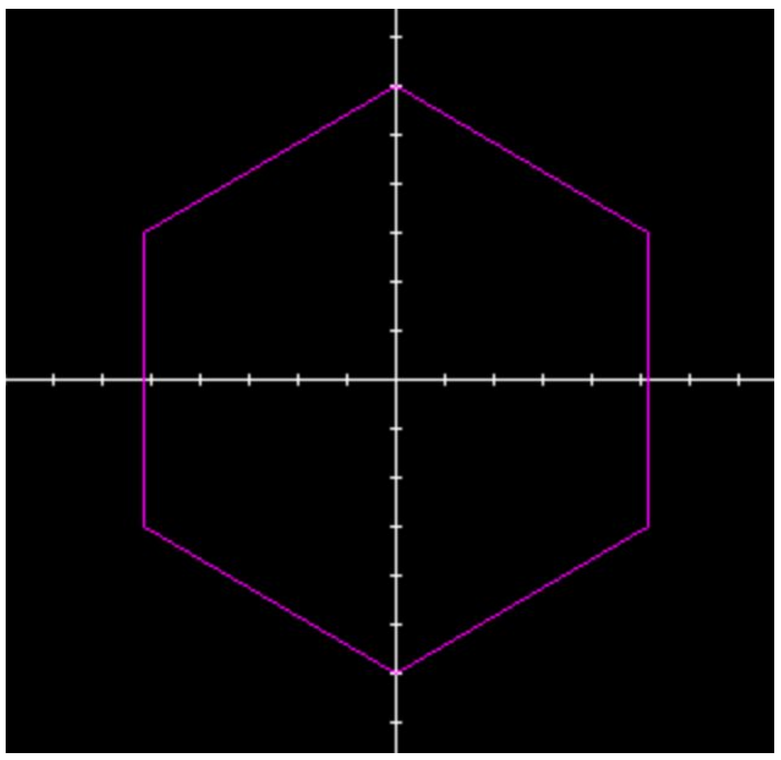
Pour les cas ou on a trois points (triangle), on a créé une fonction qui nous permet de savoir le type du triangle, on s'est basé généralement sur les longueur des côtes (segments) pour le type isocèle et équilatéral, et sur le théorème de PYTHAGOR pour le type rectangle.

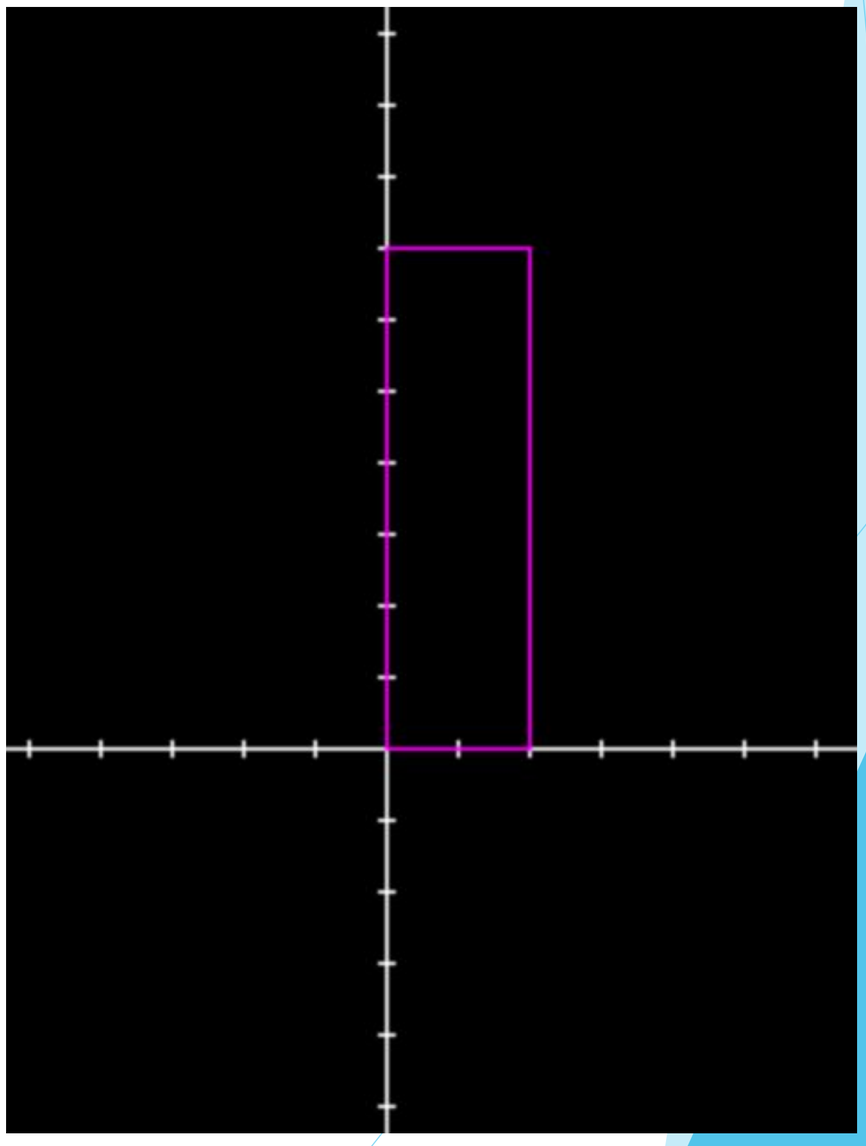
- ***Traitement des Quadrilateres (size = 4);***

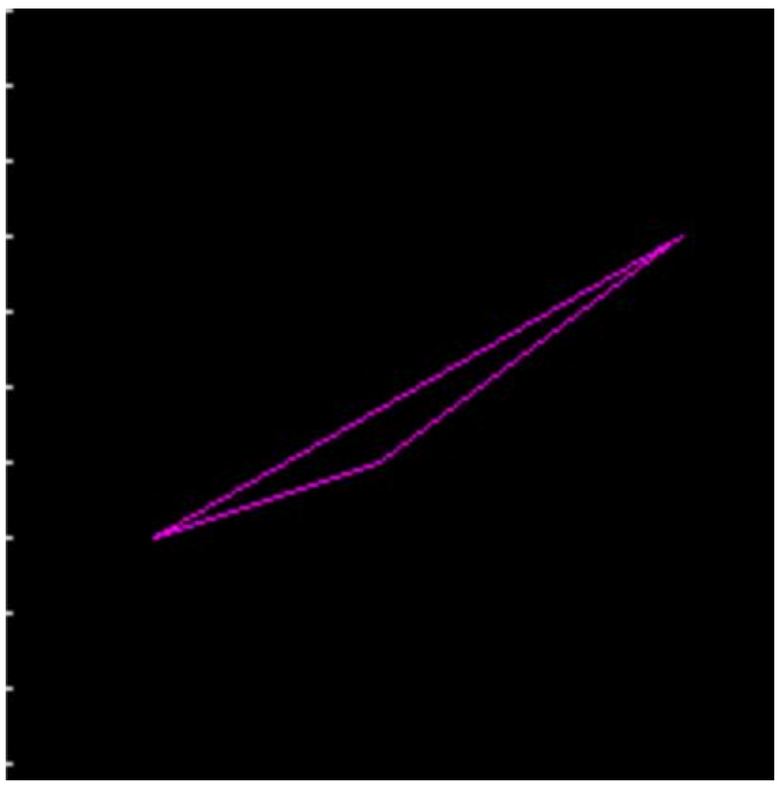
Pour les cas ou on a quatre points (quadrilatères), on a créé une fonction qui nous permet de savoir le type du polygon. La premiere chose a faire pour size = 4 est de calculer les tendons. La deuxième est de réaliser une fonction qui calcule le produit scalaire de deux vecteurs (4 points).
A l'aide du produit scalaire on vérifie si les côtes face à face sont parallèles, et si les tendons sont perpendiculaire, en plus de les longueurs des côtés nous pouvons déterminer le type du quadrilatère; Carré, Losange, Rectangle, Parallélogramme ou Trapèze ou XX pour les formes indéterminées.

- ***Dessin du Polygone (Bibliotheque mlx.h);***

Après la saisie des points a partir du fichier a une liste chaînée. A l'aide des fonctions prédéfinies dans mlx on crée une fenêtre, puis une image (qui sera collé dans la fenêtre). On trace un repère dans notre image, puis on pose les points dans ce repère. A l'aide d'une autre fonction on peut dessiner les lignes entre ces points.







3- Conception de l'application

A - Les structures de données

```c
typedef struct s_point
{
    float x;
    float y;
}           t_point;

typedef struct s_lst
{
    t_point         data;
    char             name;
    struct s_lst *next;
}               t_lst;

typedef struct s_polygone
{
    t_point     *pnts;
    int         size;
    float       *segments;
    float       *tendon;
    float       *angles;
    char        *type;
}              t_polygone;
```

struct t_point: le stockage des coordonnées d'un point.

struct t_lst : une liste chaînée pour le stockage des points d'un polygone;
Chaque maillon contient le nom du point et une structure t_point contenant ces coordonnées.

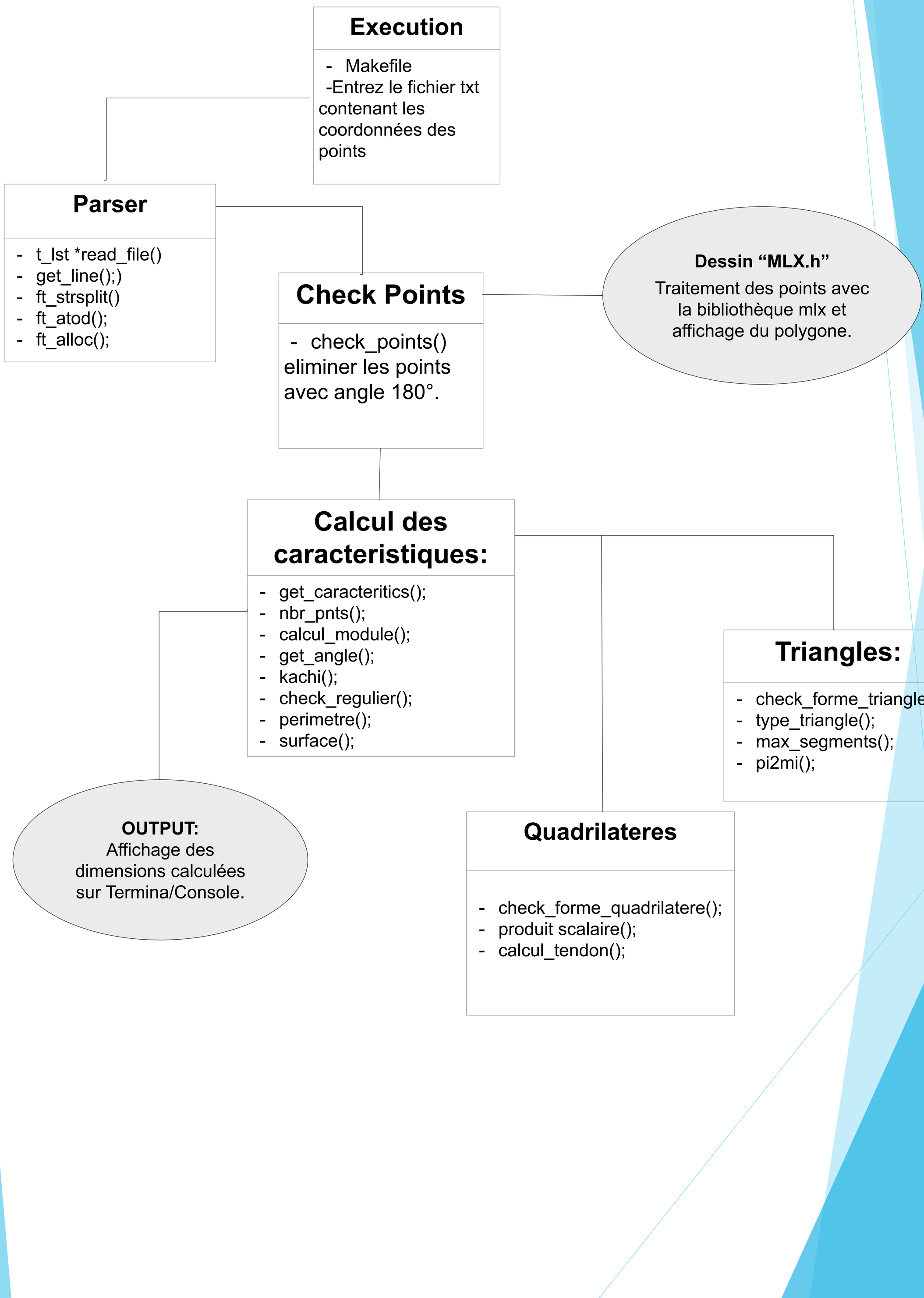
struct t_polygone : Une structure contenant la liste des points, et les différents constituants du polygone (nombre de sommets, les segments, les angles et le type).

B - Le fichier Header: polygone.h

Le fichier polygone.h est le header contenant les prototypes des différentes fonctions utilisées, et les bibliothèques nécessaire pour la réalisation de notre programme plus que les structures utilisées..

```c
# ifndef POLYGONE_H
# define POLYGONE_H

# include <stdlib.h>
# include <string.h>
# include <math.h>
# include <stdio.h>
# include <mlx.h>

#define PI 3.14159265358979323846

//**********Parser && Check points*****************
double      ft_atod(char *str);
t_lst       *create_args(void);
t_lst       *read_file(char *av);
t_lst       *check_points(t_lst **l, t_polygone *polygone);
t_point     *ft_points(int size, t_lst *l);
t_lst         *check_points(t_lst **lst, t_polygone *polygone);

//***********Les Caracteristiques****************
int         nbr_points(t_lst **l);
void        calcul_module(t_lst *l, t_polygone *polygone);
double      max_segments(t_polygone *polygone);
double      perimetre(t_polygone *polygone);
float       surface(t_polygone* polygone);
void        get_angle(t_polygone *polygone);
int         check_regulier(t_polygone *polygone);

//**********Triangeles && Quadrilateres***************
char        *check_forme_triangle(t_polygone *polygone);
char        *check_forme_quadrilatere(t_polygone *polygone);
void        calcul_tendon(t_polygone *polygone, t_lst *l);

int         pi2mi(float seg[3], float max);
int         produit_scalaire(t_point p0, t_point p1, t_point p3, t_point p2);
float       kachi(t_point A, t_point B, t_point C);

//**********Affichage*****************
void        display_all(t_polygone *polygone);
void        display_caracteristics(t_polygone *polygone);
void        display_seg_ang(t_polygone *polygone);
void        dessin_mlx(t_lst *lst);
```

C- Organigramme du Projet



D - Les fonctions de base :

Parser:

La fonction **read_file**:

Cette fonction nous permet de lire un fichier txt, ligne par ligne (get_line) et les stocke en chaîne de caractères qu'on va traiter avec avec ft strsplit pour découper chaque ligne, puis on alloue maillon par maillon (create_args)et on remplit notre liste

```
t_lst           *read_file(char *av)
{
    t_lst   *l;
    t_lst   *tt;
    t_lst   *curr;
    int     fd;
    char    *line;
    char    **splitted;

    l = NULL;
    tt = NULL;
    curr = NULL;
    if ((fd = open(av, O_RDONLY)) < 0)
        return (NULL);

    curr = create_args();
    while (get_line(fd, &line) > 0)
    {
        splitted = ft_strsplit(line, ' ');
        data.x = ft_atod(splitted[0]);
        data.y = ft_atod(splitted[1]);
        if (l == NULL)
        {
            l = create_args();
            tt = l;
            l->data = curr->data;
        }
        else
        {
            l->next = create_args();
            l->next->data = curr->data;
            l = l->next;
        }
    }
    free(curr);
    l = tt;
    return (l);
}
```

Prototype	**int get_line(int fd, char **line) ;**
Description	Une fonction vous permettant de lire une ligne terminée par un retour à la ligne depuis un file descriptor.
IN. #1	Le file descriptor du fichier
IN. #2	Une chaîne de caractère avec le contenu du ligne de fd
OUT	1 en cas de reussite; 0 en eof; -1 en cas d'erreur
Complexité	O(n) :

Prototype	**char ** ft_strsplit(char *str, char c);**
Description	Alloue (avec malloc(3)) et retourne un tableau de chaînes de caractères "fraiches" (toutes terminées par un '\0', le tableau également donc) résultant de la découpe de str selon le caractère c. Si l'allocation echoue, la fonction retourne NULL. Exemple : ft_strsplit("*salut*les***étudiants*", '*') renvoie le tableau ["salut", "les", "etudiants"].
IN. #1	La chaîne de caractères à découper.
IN. #2	Le caractère selon lequel découper la chaîne.
OUT	Le tableau de chaînes de caractères "fraîches" résultant de la découpe.
Complexité	O(n) : dépend de la taille de str, et du nombre d'occurrence de c.

Prototype	**double ft_atod(char *str) ;**
Description	Convertir une chaine de caractère en un réel.
IN. #1	La chaine de caractère à convertir.
OUT	Le Nombre reel.
Complexité	O(n) :

Check points:

```
t_lst         *check_points(t_lst **lst, t_polygone *polygone)
{
    t_lst *tete;
    t_lst *curr;
    t_lst  *lista;

    t_lst   *l;

    int i = 0;

    l = *lst;
    lista = NULL;

    curr = create_args();
    curr->name = '0';
    if((int)(polygone->angles[polygone->size - 1] * 180/PI) == 180)
        l = l->next;
    else
    {
        curr->data = l->data;
        curr->name = 'A';
        ft_alloc(&lista, &curr, &tete);
        l = l->next;
    }
    while(i < polygone->size - 1)
    {
        if ((int)(polygone->angles[i] * 180/PI) == 180)
        {
            i++;
            l = l->next;
        }
        else
        {
            if(curr->name == '0')
                curr->name = 'A';
            else
                curr->name = curr->name + 1;
            curr->data = l->data;
            ft_alloc(&lista, &curr, &tete);
            l = l->next;
            i++;
        }
    }
    free(curr);
    lista = tete;
    return (lista);
}
```

La fonction **check_points:** prends en paramètre la liste déjà avec read_file, puis on vérifie tous les angles et en cas ou l'angle est 180° on ne le stocke pas dans notre nouvelle liste retournée (t_lst *lista)

Calcul des Caracteristiques:

-La fonction **kachi** est une implémentation du théorème mathématiques d'AL KACHI.

```
float    kachi(t_point A, t_point B, t_point C)
{
    float    AB, AC, BC;
    float    k;

    AB = powf((B.x - A.x), 2)+ powf((B.y - A.y), 2);
    AC = powf((C.x - A.x), 2)+ powf((C.y - A.y), 2);
    BC = powf((C.x - B.x), 2)+ powf((C.y - B.y), 2);

    k = (AB + BC - AC) / (2 * sqrtf(AB) * sqrtf(BC));
    return (k);
}

void    get_angle(t_polygone *polygone)
{
    float *angle;
    int i = 0;
    t_point *tab;
    float k;

    tab = polygone->pnts;

    angle = (float *)malloc(sizeof(float) * polygone->size);
    polygone->angles = angle;
    while (i < polygone->size - 2)
    {
        k = kachi(tab[i], tab[i+1], tab[i+2]);
        angle[i] = acos(k);

        i++;
    }
    k = kachi(tab[i], tab[i+1], tab[0]);
    angle[i] = acos(k);
    k = kachi(tab[i+1], tab[0], tab[1]);
    angle[i + 1] = acos(k);
}
```

La fonction **get_angle:** calcule les angles des polygones en parcourant la liste des coordonnées et à l'aide de la fonction kochi. On stocke ces angles en un pointeur sur *float* (polygone->angles).

```c
float   surface(t_polygone* polygone)
{
    float   s = 0;
    int     i = 0;
    t_point *pnts;

    pnts = polygone->pnts;
    while(i < polygone->size - 1)
    {
        s = s + (pnts[i].x * pnts[i+1].y - pnts[i+1].x * pnts[i].y);
        i++;
    }
    s = s + (pnts[i].x * pnts[0].y - pnts[0].x * pnts[i].y);
    return (fabs(s / 2));
}
```

Fonction **Surfac:e** permet de calculer la surface d'un polygon à partir des coordonnées des points.

Prototype	**void calcul_modules(t_lst *lst, t_polygone *polygone) ;**
Description	Calcule des distances des côtes du polygone; on alloue un pointeur (polygone->size) et on stocke les résultats dans le pointeurs sur *float* polygone->segments.
IN. #1	La liste des coordonnées des points
IN. #2	structure polygone
OUT	void
Complexité	O(n) :

La relation pour calculer la distance $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Prototype	**int nbr_points(t_lst **lst) ;**
Description	Calculer le nombre des points (sommets); la longueur de la liste chaînée.
IN. #1	La liste contenant les coordonnées des points.
OUT	Le nombre de maillons du liste.
Complexité	O(n) :

Prototype	**double perimetre(t_polygone *polygone);**
Description	Calculer le périmètre du polygone, en sommant les distances des segments (polygone->segments)
IN. #1	La structure du polygone.
OUT	Le perimetre du polygone .
Complexité	O(n) :

Prototype	**int check_regulier(t_polygone *polygone);**
Description	On compare les segments et les angles du polygone (polygone->segments / polygone->angles).
IN. #1	La structure du polygone.
OUT	1 si le polygone est regulier, 0 sinon.
Complexité	O(n) :

Triangles:

```c
char        *check_forme_triangle(t_polygone *polygone)
{

    float max;
    max = 0;

    max = max_segments(polygone);
    printf("Type(Triangle):");
    if (pi2mi(polygone->segments, max) == 1 && type_triangle(polygone) == 1 )
        printf("\tIsocele Rectangle\n");
    else if (pi2mi(polygone->segments, max) == 1)
        printf("\tRectangle\n" );
    else if (type_triangle(polygone) == 1)
       printf("\tIsocele\n");
    else if (type_triangle(polygone) == 2)
        printf("\tEquilaterale\n");
    else
        printf("\tXX \n");
  return ("XX");
}
```

La fonction **check_forme_triangulaire** précise le type du triangle (Isocele, equilateral, rectangle, rectangle isocele).

Prototype	**double max_segments(t_polygone *polygone);**
Description	On cherche le maximum des segments du polygone.
IN. #1	La structure du polygone(polygone->segments).
OUT	max
Complexité	O(n) :

```c
int     pi2mi(float segments[3], float max)
{
  float i;
  float j;
  float k;
  i = segments[0];
  j = segments[1];
  k = segments[2];
    if((i == max) && (ceil(i*i) == ceil(j*j + k*k)))
      return (1);
    if ((j == max) && (j*j == i*i + k*k))
      return (1);
    if ((k == max) && (k*k == i*i + j*j))
      return (1);

return (0);

}
```

Cette fonction vérifie si un triangle est rectangle ou pas (1 ou 0) à l'aide du théorème PYTHAGORE.

```c
int     type_triangle(t_polygone *polygone)
{
  double i;
  double j;
  double k;

  i = polygone->segments[0];
  j = polygone->segments[1];
  k = polygone->segments[2];
   if ((i == j && i != k) || (i == k && i != j) ||(i != j && j == k))
      return (1);
    if(i == j && j == k && k == i)
      return (2);

   return (0);
}
```

Cette fonction vérifie si un triangle est isocele 1 ou équilatéral 2 ou 0 sinon.

Quadrilateres:

```
char    *check_forme_quadrilatere(t_polygone *polygone)
{
    t_point *pnts;
    float ps;
    float ps1;
    float ps_td;

    pnts = polygone->pnts;
    ps = abs(produit_scalaire(pnts[0], pnts[1], pnts[2], pnts[3]));
    ps1 = abs(produit_scalaire(pnts[1], pnts[2], pnts[0], pnts[3]));
    ps_td = abs(produit_scalaire(pnts[0], pnts[2], pnts[1], pnts[3]));
    /*************
        ps et ps1 sont les produits scalaires des cote face a face
        ps_td est le produit scalaire des tendons
                                        ******************/
    if ((ps == polygone->segments[0] * polygone->segments[2])
            && (ps1 == polygone->segments[1] * polygone->segments[3]))
    {
        if ((polygone->segments[0] == polygone->segments[2])
                && (polygone->segments[1] == polygone->segments[3]))
        {
            if (ps_td == 0)
            {
                if(polygone->tendon[0] == polygone->tendon[1])
                    return("Carre.");
                else
                    return ("Losange.");
            }
            else if (polygone->tendon[0] == polygone->tendon[1])
                return ("Rectangle.");
            else
                return("Parallélogramme.");
        }
        else
            return ("XX");
    }
    else if ((ps == polygone->segments[0] * polygone->segments[2])
            || (ps1 == polygone->segments[1] * polygone->segments[3]))
        return ("Trapeze.");
    else
        return ("XX");
}
```

La fonction **check_forme_quadrilateres** vérifie la nature du polygone à 4 sommets.

On vérifie si les cotes qui sont face à face sont parallèles, et si les tendons sont rectangulaire, en plus de la comparaison des distances des côtes du polygones, afin de savoir la nature du quadrilat eres: Carré, Rectangle, Losange, Parallélogramme ou Trapèze.

Prototype	**double produit scalaire(t_point A, t_point B, t_point C, t_point D);**
Description	Calculer le produit scalaire AB.CD ç l'aide des coordonnées des quatres points. pour verifier si les 2 vecteurs sont parrallele ou rectangulaire.
IN. #1 / IN. #2 IN. #3 / IN. #4	Les structure contenant les coordonnées des points.
OUT	Le produit scalaire.
Complexité	O(n) :

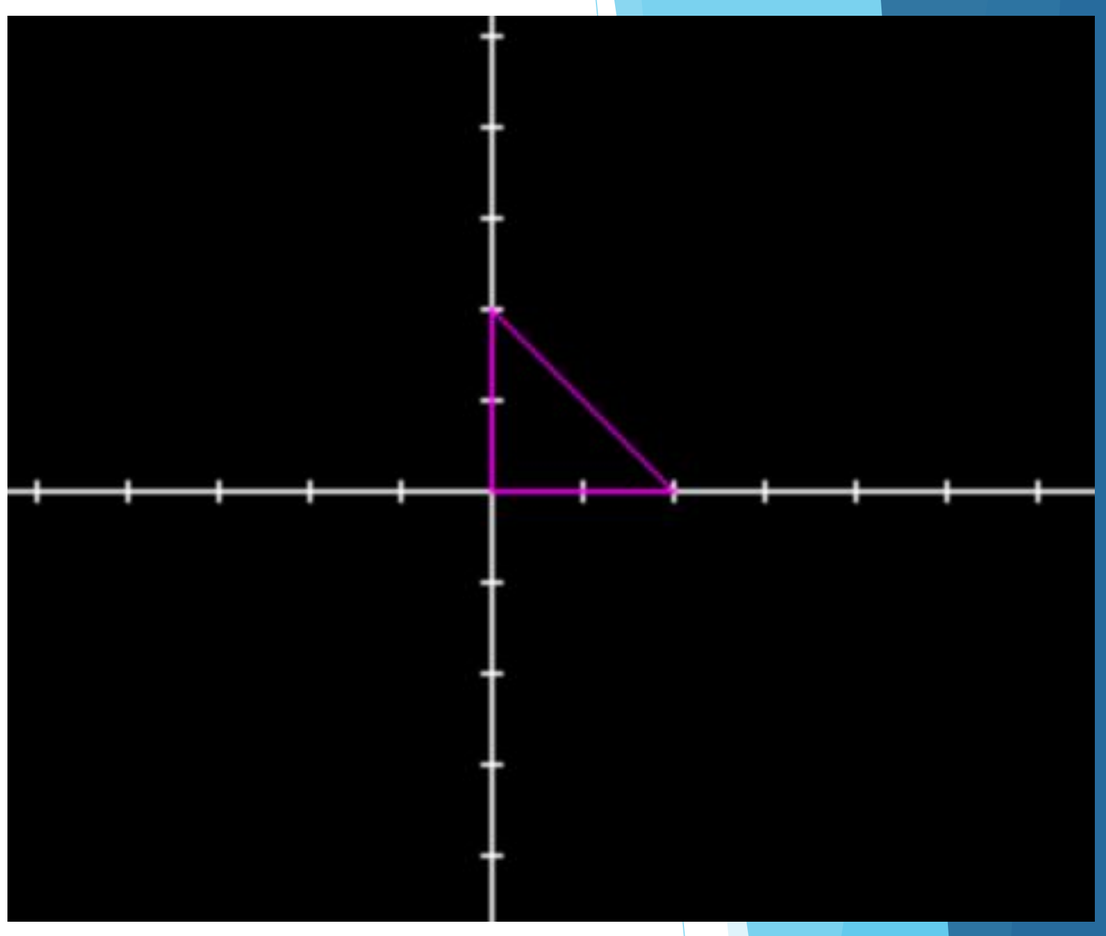
MLX:

```
void        dessin_mlx(t_lst *lst)
{

    p->mlx = mlx_init();
    p->win = mlx_new_window(p->mlx, WIDTH, HEIGHT, title);
    p->img = mlx_new_image(p->mlx, WIDTH, HEIGHT);
    p->pixels = (int *)mlx_get_data_addr(p->img, &p->bpp, &p->size_line, &p->endian);
    draw(&p);
    mlx_hook(p->win, 17, (1L << 17), prog_close, &p);
    mlx_hook(p->win, 2, 0, keys, &p);
    mlx_loop(p->mlx);
}
```

TRIANGLES:

```
*****Les Points:*****
Size: 3
Type(Triangle): Isocele Rectangle

A (x.2.00; y.0.00)
B (x.0.00; y.2.00)
C (x.0.00; y.0.00)

*****Les Segements(cotes):*****
AB = 2.83
BC = 2.00
CA = 2.00

*****Les Angles:*****
 B = 45.00
 C = 90.00
 A = 45.00

*****Les Caracteristiques::*****

Perimetre:      6.83
Surface:        2.00
Regulier?       0

*********************************
```

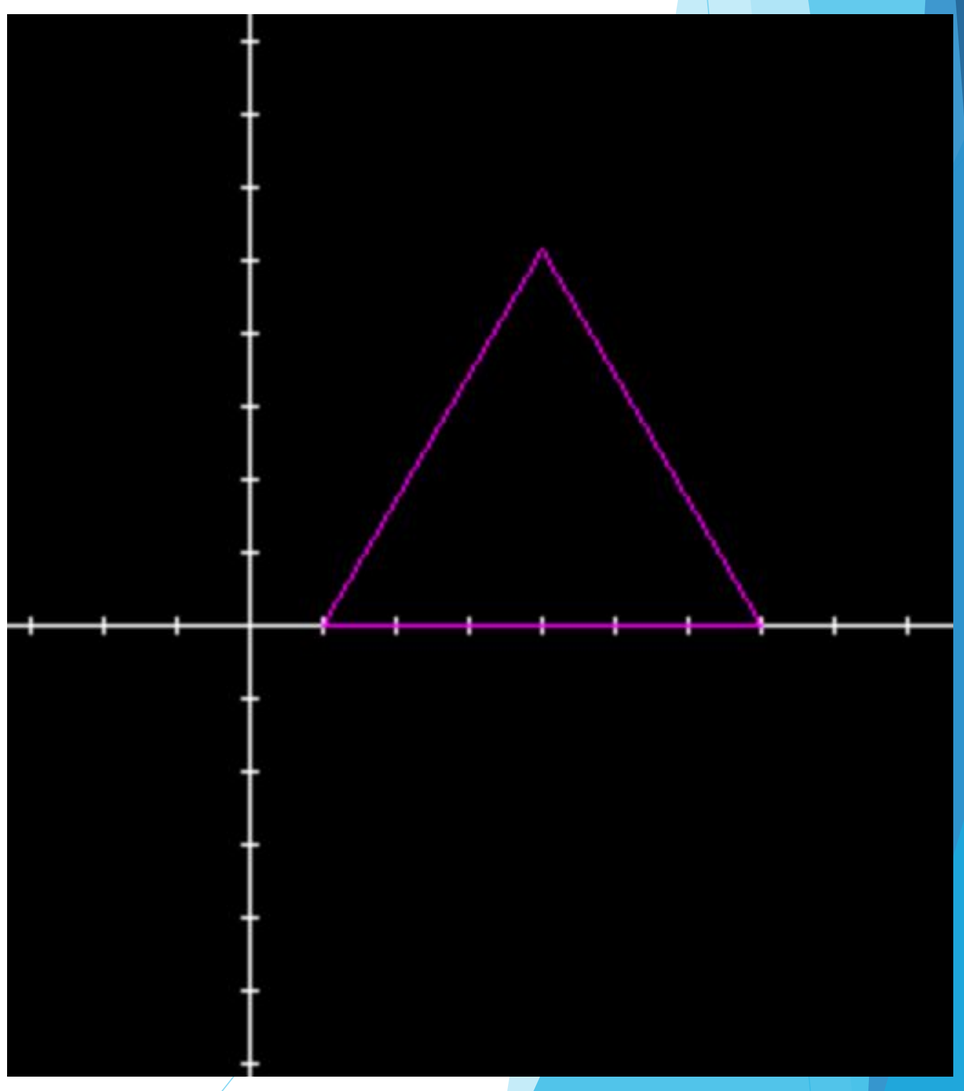
```
1 0
2 0
0 2
0 1
0 0
```



Dans cet exemple on a fait entrer 5 points avec 2 angles de 180°, notre programme n'as pris que les 3 premiers points définissant un triangle.

```
*****Les Points:*****
Size: 3
Type(Triangle): Equilaterale

A (x.1.00; y.0.00)
B (x.7.00; y.0.00)
C (x.4.00; y.5.20)

*****Les Segements(cotes):*****
AB = 6.00
BC = 6.00
CA = 6.00

*****Les Angles:*****
 B = 60.02
 C = 59.96
 A = 60.02

*****Les Caracteristiques::*****

Perimetre:      18.01
Surface:        15.60
Regulier?       1

*********************************
```

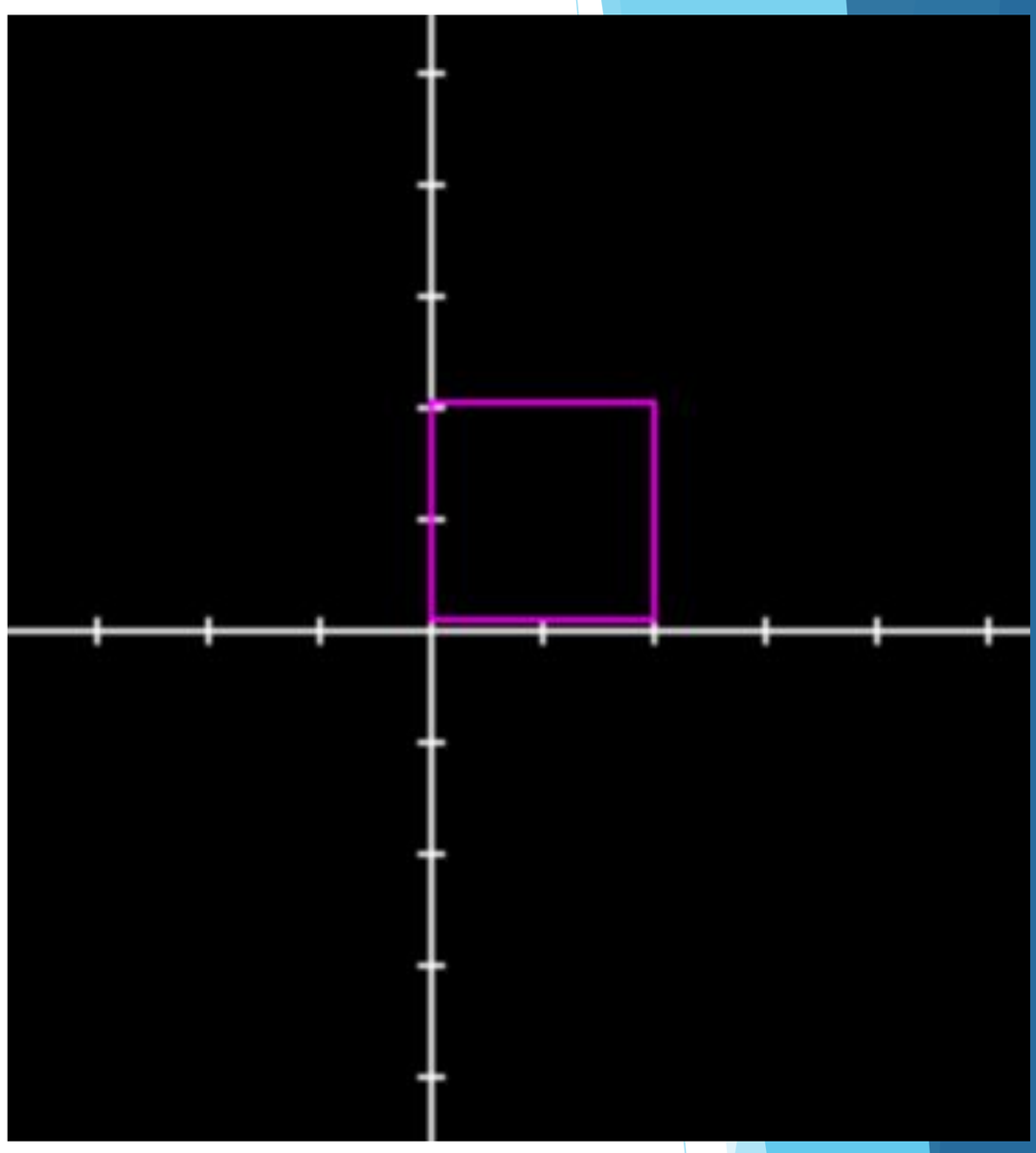
```
1   0
7   0
4   5.2
```



Quadrilateres:

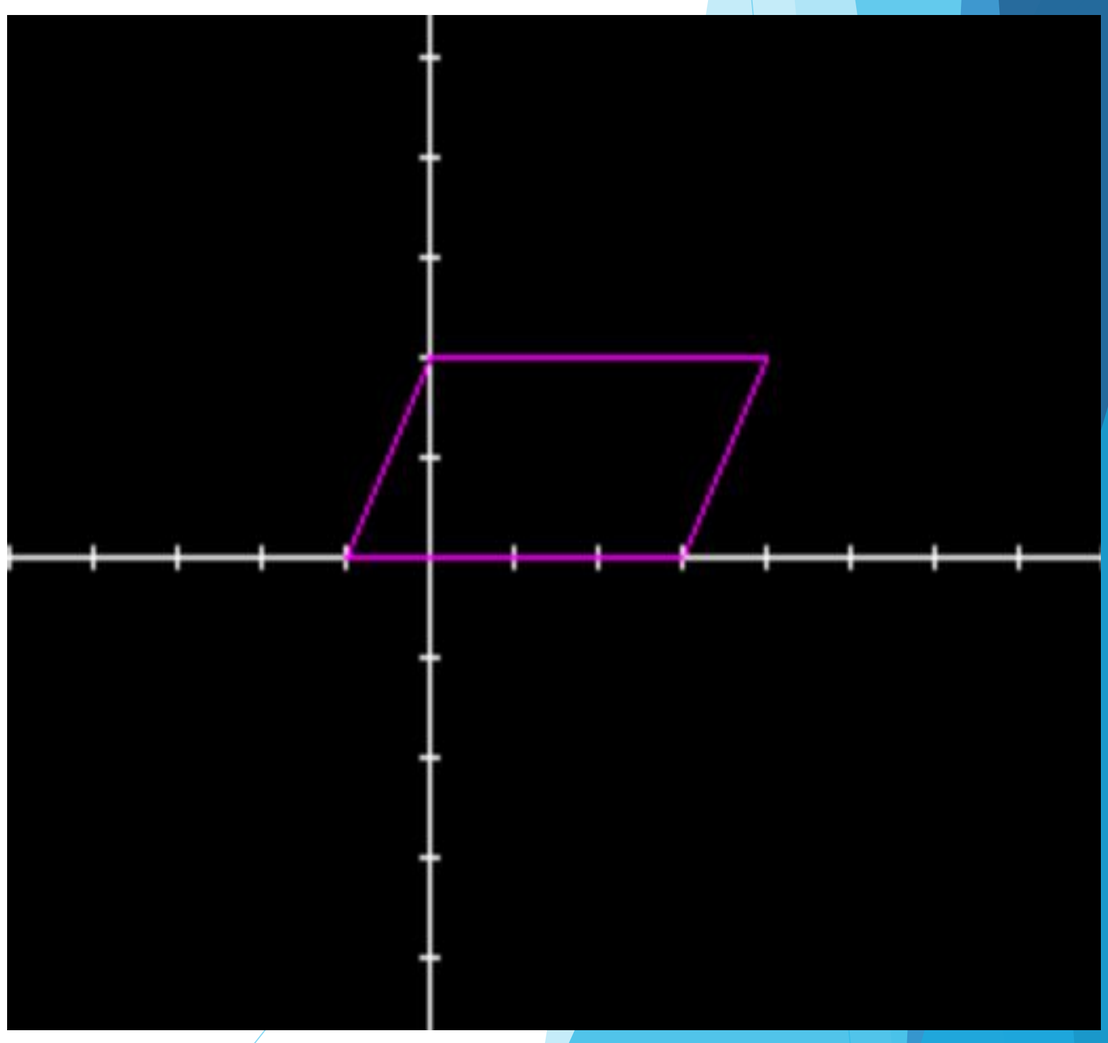
```
*****Les Points:*****
Size: 4
Type: (Quadrilatere)    Carre.

A (x.2.00; y.0.00)
B (x.2.00; y.2.00)
C (x.0.00; y.2.00)
D (x.0.00; y.0.00)

*****Les Segements(cotes):*****
AB = 2.00
BC = 2.00
CD = 2.00
DA = 2.00

*****Les Angles:*****
 B = 90.00
 C = 90.00
 D = 90.00
 A = 90.00

*****Les Caracteristiques::*****

Perimetre:      8.00
Surface:        4.00
Regulier?       1

*********************************
```

```
2 0
2 2
0 2
0 0
```



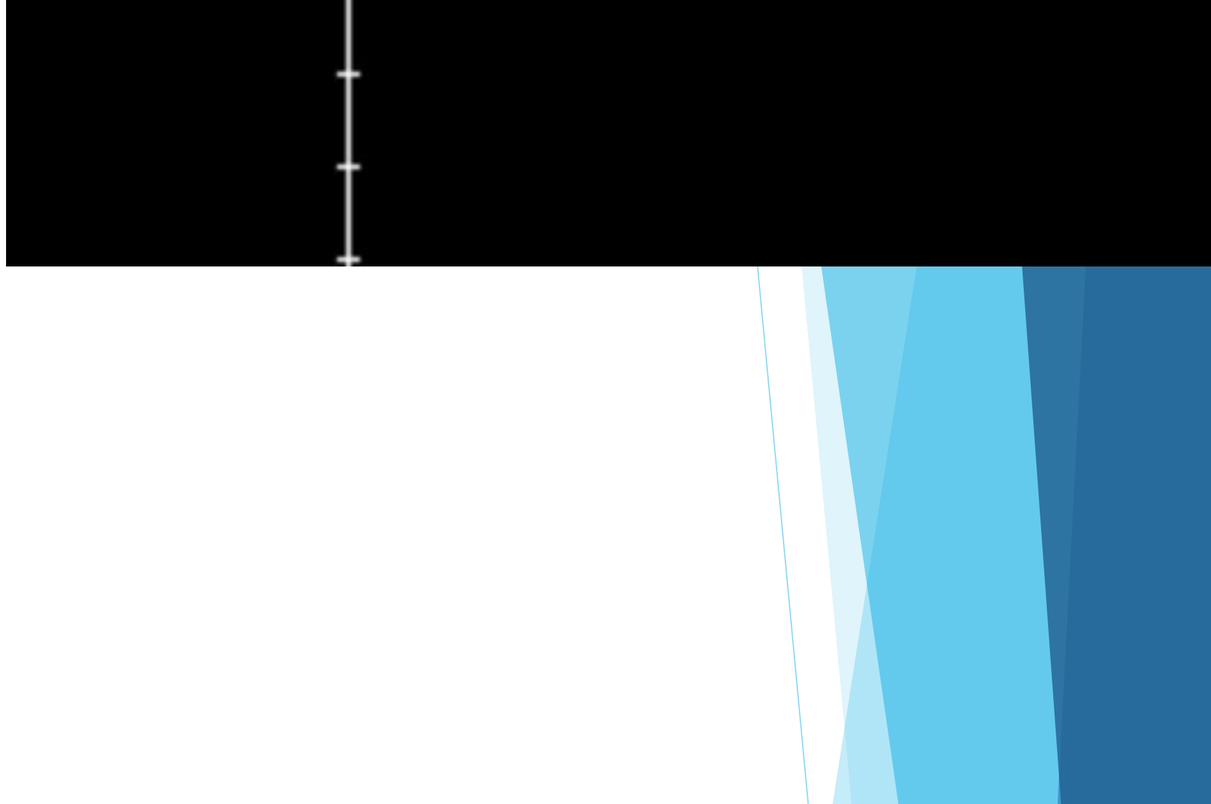
```
*****Les Points:*****
Size: 4
Type: (Quadrilatere)    Parallélogramme.

A (x.0.00; y.2.00)
B (x.4.00; y.2.00)
C (x.3.00; y.0.00)
D (x.-1.00; y.0.00)

*****Les Segements(cotes):*****
AB = 4.00
BC = 2.24
CD = 4.00
DA = 2.24

*****Les Angles:*****
 B = 63.43
 C = 116.57
 D = 63.43
 A = 116.57

*****Les Caracteristiques::*****

Perimetre:      12.47
Surface:        8.00
Regulier?       0

*********************************
```

```
0 2
4 2
3 0
-1 0
```



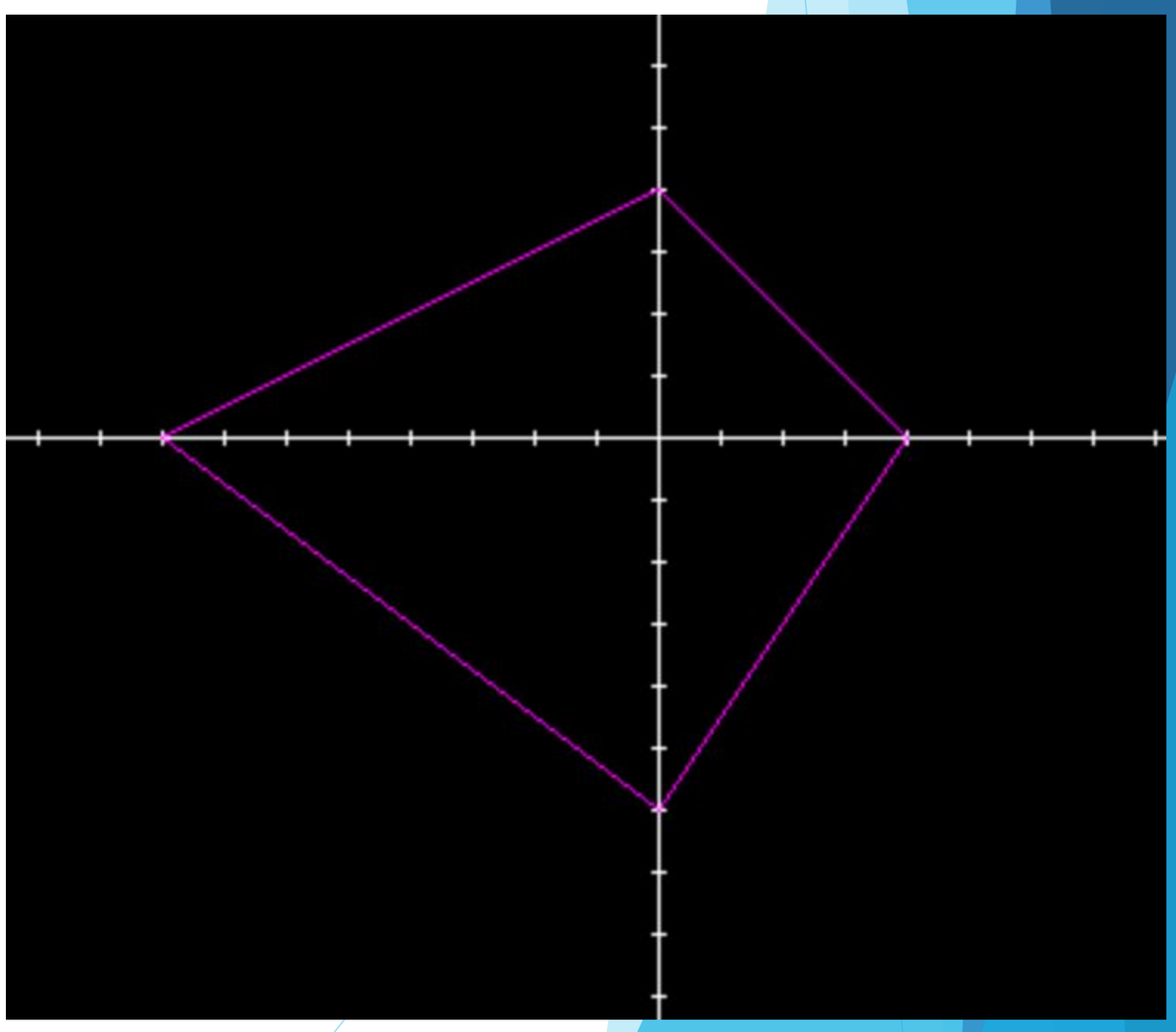
```
*****Les Points:*****
Size: 4
Type: (Quadrilatere)    Trapeze.

A (x.0.00; y.0.00)
B (x.2.00; y.2.00)
C (x.5.00; y.2.00)
D (x.8.00; y.0.00)

*****Les Segements(cotes):*****
AB = 2.83
BC = 3.00
CD = 3.61
DA = 8.00

*****Les Angles:*****
 B = 135.00
 C = 146.31
 D = 33.69
 A = 45.00

*****Les Caracteristiques::*****

Perimetre:      17.43
Surface:        11.00
Regulier?       0

**********************************
```

```
0 0
2 2
5 2
8 0
```



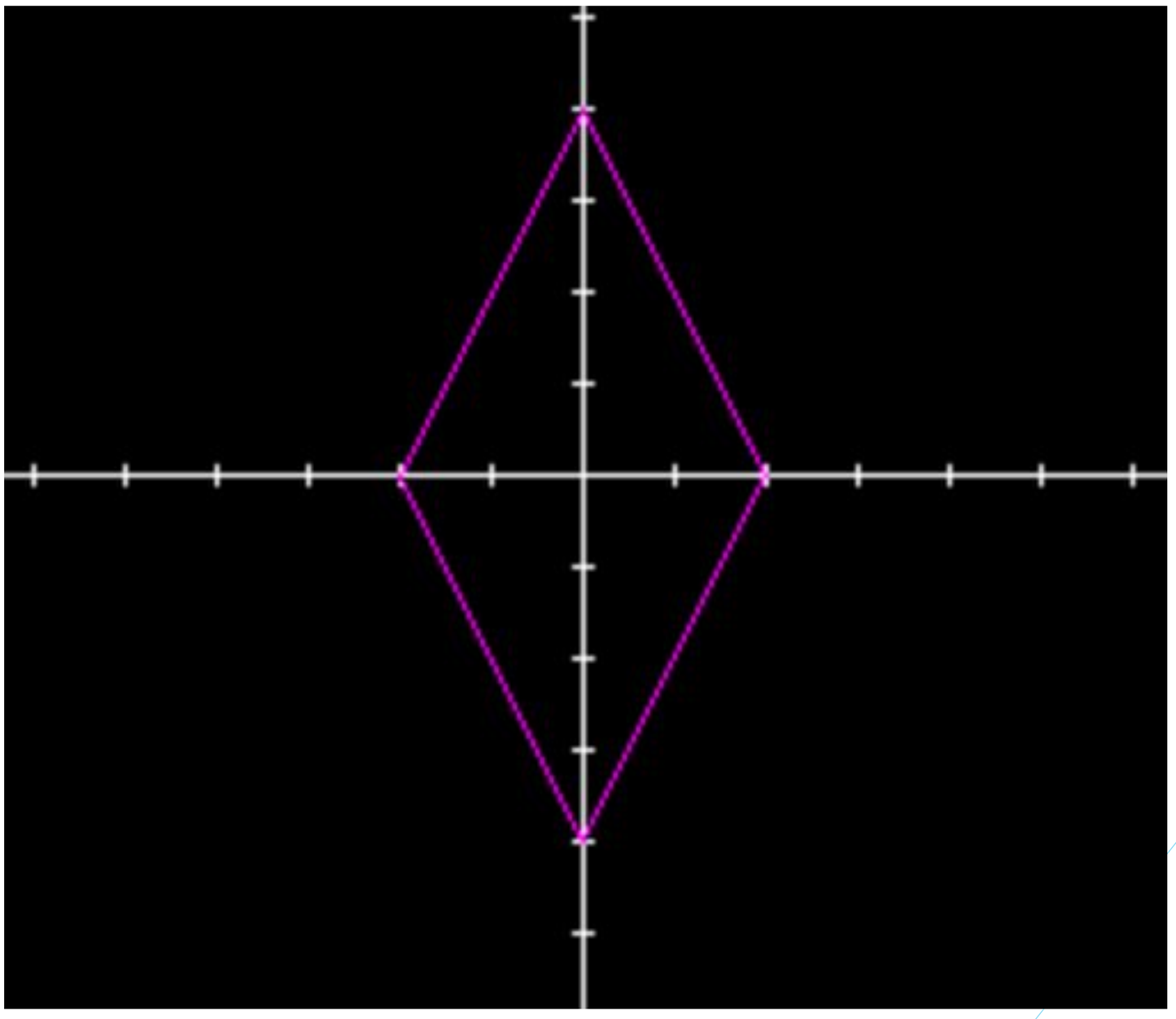
```
*****Les Points:*****
Size: 4
Type: (Quadrilatere)    XX

A (x.0.00; y.2.00)
B (x.2.00; y.0.00)
C (x.0.00; y.-3.00)
D (x.-3.00; y.0.00)

*****Les Segements(cotes):*****
AB = 2.83
BC = 3.61
CD = 4.24
DA = 3.61

*****Les Angles:*****
 B = 101.31
 C = 78.69
 D = 78.69
 A = 101.31

*****Les Caracteristiques::*****

Perimetre:      14.28
Surface:        12.50
Regulier?       0

**********************************
```

```
 0   2
 2   0
 0  -3
-3   0
```



```
*****Les Points:*****
Size: 4
Type: (Quadrilatere)    Losange.

A (x.0.00; y.2.00)
B (x.1.00; y.0.00)
C (x.0.00; y.-2.00)
D (x.-1.00; y.0.00)

*****Les Segements(cotes):*****
AB = 2.24
BC = 2.24
CD = 2.24
DA = 2.24

*****Les Angles:*****
 B = 126.87
 C = 53.13
 D = 126.87
 A = 53.13

*****Les Caracteristiques::*****

Perimetre:      8.94
Surface:        4.00
Regulier?       0

**********************************
```

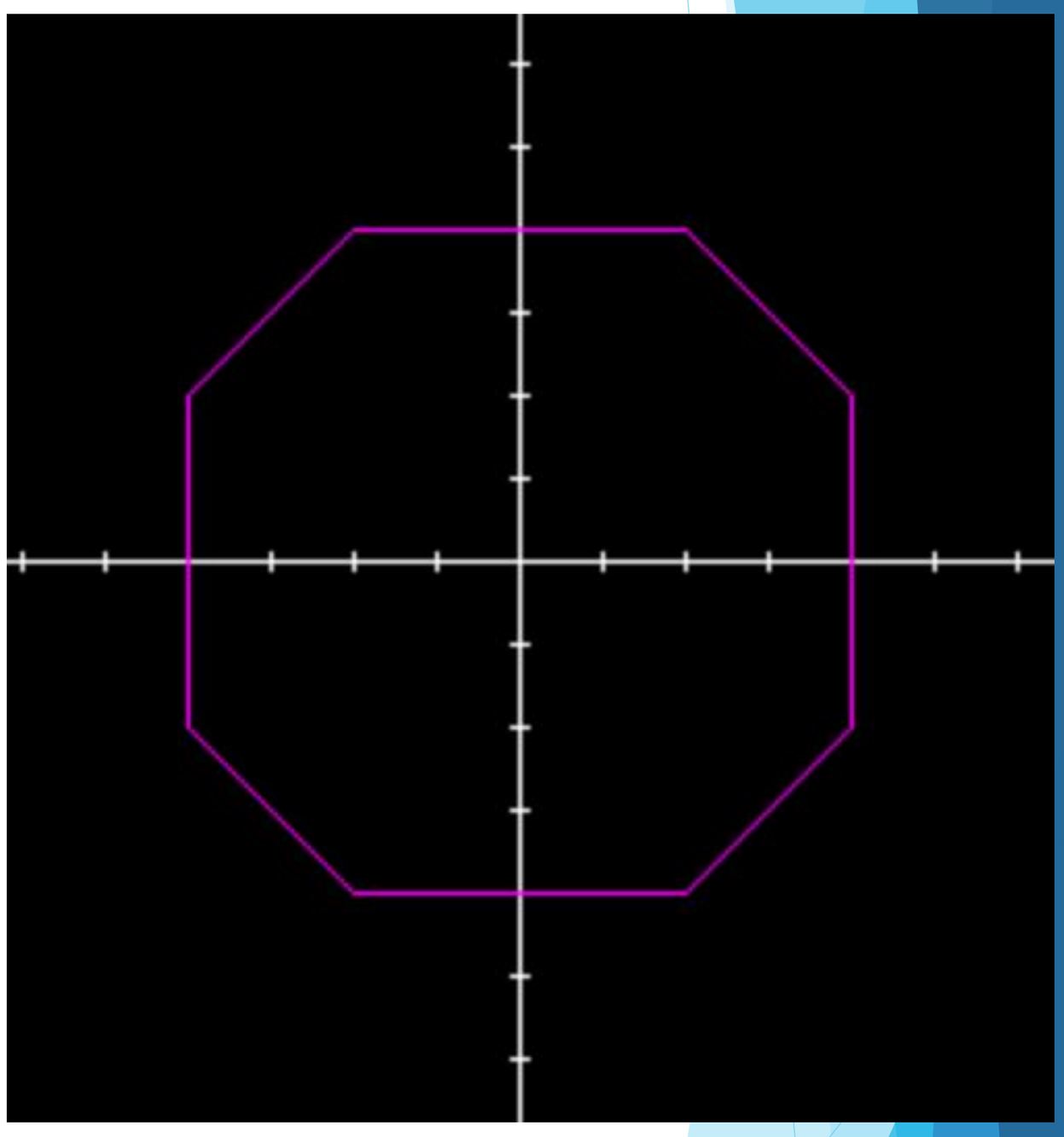
```
0 2
1 0
0 -2
-1 0
```



Autres Exemple:

```
*****Les Points:*****
Size: 8
Type:   XX

A (x.2.00; y.4.00)
B (x.4.00; y.2.00)
C (x.4.00; y.-2.00)
D (x.2.00; y.-4.00)
E (x.-2.00; y.-4.00)
F (x.-4.00; y.-2.00)
G (x.-4.00; y.2.00)
H (x.-2.00; y.4.00)

*****Les Segements(cotes):*****
AB = 2.83
BC = 4.00
CD = 2.83
DE = 4.00
EF = 2.83
FG = 4.00
GH = 2.83
HA = 4.00

*****Les Angles:*****
 B = 135.00
 C = 135.00
 D = 135.00
 E = 135.00
 F = 135.00
 G = 135.00
 H = 135.00
 A = 135.00

*****Les Caracteristiques::*****

Perimetre:      27.31
Surface:        56.00
Regulier?       0

**********************************
```

```
 2    4
 4    2
 4   -2
 2   -4
-2   -4
-4   -2
-4    2
-2    4
```



Conclusion

Notre programme consiste à réaliser un polygone et calculer tous ces caractéristiques et préciser son type.

Nous avons pu mettre en œuvre et appliquer toutes les connaissances que nous avons accumulées durant cette semestre, ce travail nous a même poussés à chercher de nouvelles connaissances.

Apres plusieurs tests et correction nous avons pu finaliser et mettre notre programme en œuvre et en bon fonctionnement, en espérant faire un travail présentable.

Nous espérons que nous étions à la hauteur de vos attentes et que nous avons bien suivi vos consignes.