I Objectifs du TD

1) Ecrire plusieurs algorithmes concernant la géométrie plane :

- Déterminer et afficher le milieu d'un segment
- Déterminer et afficher le quatrième sommet d'un parallélogramme
- Calculer la distance entre deux points
- Déterminer si un triangle donné est isocèle

2) Implémenter ces différents algorithmes sur ordinateur avec Xcas.

Pour aller plus loin :

1) Généraliser l'algorithme qui détermine si un triangle est isocèle afin d'établir la nature d'un triangle (équilatéral, isocèle, isocèle rectangle, rectangle ou quelconque).

II Milieu d'un segment

a) Algorithme

Proposer un algorithme qui demande les coordonnées de deux points A et B et calcule les coordonnées du point I milieu de A et B.

b) Implémenter cet algorithme sous Xcas

On pourra utiliser les instructions M:= point(a,b) qui affiche le point

M(a;b) et segment(A,B) qui trace le segment [AB].

Pour afficher la figure tracée, il faut faire : Cfg/Montrer/Montrer DispG.

III Quatrième sommet d'un parallélogramme

a) Algorithme

Proposer un algorithme qui demande les coordonnées de trois points A, B, C et calcule les coordonnées du point D tel que ABCD soit un parallélogramme.

Afficher le parallélogramme ABCD.

b) Implémenter cet algorithme sous Xcas

On pourra créer une fonction qui entrée a comme paramètres les coordonnées de deux points et retourne la distance entre ces deux points.

IV Triangle isocèle

a) Algorithme

Proposer un algorithme qui utilise la fonction précédente qui calcule la distance entre deux points pour déterminer si un triangle est isocèle.

On pourra tester le cas particulier d'un triangle équilatéral.

b) Implémenter cet algorithme sous Xcas

On pourra tracer le triangle.

V Triangles isocèle, rectangle et équilatéral

a) Algorithme

Proposer un algorithme qui généralise celui du paragraphe IV afin de déterminer la nature d'un triangle : isocèle, rectangle, isocèle rectangle, équilatéral ou quelconque.

b) Implémenter cet algorithme sous Xcas

On pourra tracer le triangle.

CORRECTION

II Milieu d'un segment

a) Algorithme

Milieu_Segment

Début

Variables

xA , yA , xB , yB , xC , yC , xI , yI

Entrees

Saisir *xA , yA , xB , yB ,*

Traitement

xI ***prend la valeur (****xA+xB)/2*

yI ***prend la valeur (****yA,yB)/2*

Sorties

Afficher le point A(xA,yA)

Afficher le point B(xB,yB)

Afficher le point I(xI,yI)

Tracer le segment [AB]

Fin

CORRECTION

b) Implémenter cet algorithme sous Xcas

Ouvrir une fenêtre de programmation avec ALT p.

Saisir le code suivant :

```
milieu_segment(xA,yA,xB,yB) := {
erase();
xI := (xA+xB)/2;
yI := (yA+yB)/2;
A := point(xA,yA);
legende(A,"A");
B:=point(xB,yB);
legende(B,"B");
I := point(xI,yI);
legende(I,"I");
segment(A,B);
}:;
```

Pour afficher la figure tracée, il faut faire : Cfg/Montrer/Montrer DispG.

CORRECTION

III Quatrième sommet d'un parallélogramme

a) Algorithme

Sommet_4_Parallélogramme

Début

Variables

xA , yA , xB , yB , xC , yC , xO , yO,xD,yD

Entrees

Saisir *xA , yA , xB , yB , xC, yD*

Traitement

xO ***prend la valeur*** *(xA+xC)/2*

yO ***prend la valeur*** *(yA,yC)/2*

xD ***prend la valeur*** *2×xO - xB*

yD ***prend la valeur*** *2×yO - yB*

Sorties

Afficher le point A(xA,yA)

Afficher le point B(xB,yB)

Afficher le point C(xC,yC)

Afficher le point D(xD,yD)

Afficher le point O(xO,yO)

Tracer le polygone ABCD

Fin

CORRECTION

b) Implémenter cet algorithme sous Xcas

```
le_parallelogramme(xA,yA,xB,yB,xC,yC) := {
local xO,yO,xD,yD,A,B,C,D,O;
erase();
// Calcul des coordonnées du point O milieu de [AC]
xO := (xA+xC)/2;
yO := (yA+yC)/2;
// Calcul des coordonnées du point D tel que O soit le milieu de [BD]
xD := 2*xO - xB;
yD := 2*yO - yB;
A := point(xA,yA);
legende(A,"A");
B:=point(xB,yB);
legende(B,"B");
C := point(xC,yC);
legende(C,"C");
D := point(xD,yD);
legende(D,"D");
O := point(xO,yO);
legende(O,"O");
segment(A,B);
segment(B,C);
segment(C,D);
segment(D,A);
};;
```

CORRECTION

IV Distance entre deux points

a) Algorithme

Distance_2_points

Début

Variables

xA , yA , xB , yB , dist

Entrees

Saisir *xA , yA , xB , yB*

Traitement

dist prend la valeur $\sqrt{(xB - xA)^2 + (yB - yA)^2}$

Sorties

Afficher dist

Fin

b) Implémentation sur Xcas

```
distance_deux_points(xA,yA,xB,yB) := {
  local dist;
  dist := sqrt((xB - xA)^2 + (yB - yA)^2);
  return dist;
}
:;
```

Exemple d'utilisation : *print(distance_deux_points(1,2,3,4));*

CORRECTION

IV Triangle isocèle

a) Algorithme

```
Triangle_isocèle
Début
    Variables
    xA , yA , xB , yB , xC , yC , distAB, distAC, distBC, pyth, sommet;
    Entrees
    Saisir xA , yA , xB , yB , xC, yC
    Traitement
    distance_deux_points(xA,yA,xB,yB) ➔ distAB
    distance_deux_points(xA,yA,xC,yC) ➔ distAC
    distance_deux_points(xB,yB,xC,yC) ➔ distBC
    Sorties
    Si distAB = distAC et distAB = distBC alors
            Afficher "le triangle ABC est équilatéral"
    Sinon
            Si distAB = distAC alors
                    Afficher "le triangle ABC est isocèle en A
            Sinon
                    Si distAB = distBC alors
                            Afficher "le triangle ABC est isocèle en B
                    Sinon
                            Si distAC = distBC alors
                                    Afficher "le triangle ABC est isocèle en C
                            Fin si
                    Fin si
    Fin si
    Tracer le polygone ABC
Fin
```

b) Implémentation sur Xcas

```
triangle_iso(xA,yA,xB,yB,xC,yC) := {
 local distAB,distAC,distBC;
 distAB := distance_deux_points(xA,yA,xB,yB);
 distAC := distance_deux_points(xA,yA,xC,yC);
 distBC := distance_deux_points(xB,yB,xC,yC);
 si ((distAB == distBC) et (distAB==distAC)) alors
  print("le triangle ABC est équilatéral");
 sinon
  si (distAB ==distAC) alors
     print("le triangle ABC est isocèle en A");
  sinon
   si (distAB==distBC) alors
```

CORRECTION

```
            print("le triangle ABC est isocèle en B");
        sinon
          si (distAC==distBC) alors
              print("le triangle ABC est isocèle en C");
          sinon
              print("le triangle ABC n'est ni équilatéral ni isocèle");
          fsi
        fsi
      fsi
    fsi
    erase();
    A := point(xA,yA);
    legende(A,"A");
    B := point(xB,yB);
    legende(B,"B");
    C := point(xC,yC);
    legende(C,"C");
    segment(A,B);
    segment(A,C);
    segment(B,C);
}
:;
```

CORRECTION

V Triangles isocèle, rectangle et équilatéral

c) Algorithme

```
Nature_triangle
Début
Variables
xA , yA , xB , yB , xC , yC, sommet;
Entrees
Saisir xA , yA , xB , yB , xC, yC,distAB, distAC, distBC
Traitement
distance_deux_points(xA,yA,xB,yB) ➔ distAB
distance_deux_points(xA,yA,xC,yC) ➔ distAC
distance_deux_points(xB,yB,xC,yC) ➔ distBC
Si distAB > distAC alors
    Si distAB > distBC alors
            distAB*distAB – (distAC*distAC + distBC*distBC) ➔ pyth
            "C" ➔ sommet
    Sinon
            distBC*distBC – (distAC*distAC + distAB*distAB) ➔ pyth
            "A" ➔ sommet
    Fin si
Sinon
    Si distAC > distBC alors
            distAC*distAC – (distAB*distAB + distBC*distBC) ➔ pyth
            "B" ➔ sommet
    Sinon
            distBC*distBC – (distAC*distAC + distAB*distAB) ➔ pyth
            "A" ➔ sommet
    Fin si
Sorties
Si distAB = distAC et distAB = distBC alors
    Afficher "le triangle ABC est équilatéral"
Sinon
    Si distAB = distAC alors
            Si Pyth = 0 alors
                    Afficher "le triangle ABC est isocèle rectangle en A"
            Sinon
                    Afficher "le triangle ABC est isocèle en A
            Fin si
    Sinon
            Si distAB = distBC alors
                    Si Pyth = 0 alors
                            Afficher "le triangle ABC est isocèle rectangle en B"
                    Sinon
                            Afficher "le triangle ABC est isocèle en B
```

CORRECTION

```
                Fin si
            Sinon
                Si distAC = distBC alors
                    Si Pyth = 0 alors
                        Afficher "le triangle ABC est isocèle rectangle en C"
                    Sinon
                        Afficher "le triangle ABC est isocèle en C
                    Fin si
                Sinon
                    Si pyth = 0 alors
                        Afficher "le triangle ABC est rectangle en " sommet
                    Sinon
                        Afficher "le triangle ABC est quelconque"
                    Fin si
                Fin si
            Fin si
    Fin si
    Tracer le polygone ABC
    Fin
```

d) Implémenter cet algorithme sous Xcas

```
nature_triangle(xA,yA,xB,yB,xC,yC) := {
 local distAB,distAC,distBC,pyth,le_sommet;
 distAB := distance_deux_points(xA,yA,xB,yB);
 distAC := distance_deux_points(xA,yA,xC,yC);
 distBC := distance_deux_points(xB,yB,xC,yC);
 si (distAB > distAC) alors
  si (distAB > distBC) alors
   pyth := distAB*distAB - (distAC*distAC + distBC*distBC);
   le_sommet := "C";
  sinon
   pyth := distBC*distBC - (distAC*distAC + distAB*distAB);
   le_sommet := "A";
  fsi
 sinon
  si (distAC > distBC) alors
   pyth := distAC*distAC - (distAB*distAB + distBC*distBC);
   le_sommet := "B";
  sinon
   pyth := distBC*distBC - (distAB*distAB + distAC*distAC);
   le_sommet := "A";
  fsi
```

CORRECTION

```
fsi
pyth := simplify(pyth);
si ((distAB == distBC) et (distAB==distAC)) alors
  print("le triangle ABC est équilatéral");
sinon
  si (distAB ==distAC) alors
    si (pyth==0) alors
      print("le triangle ABC est isocèle rectangle en A");
    sinon
      print("le triangle ABC est isocèle en A");
    fsi
  sinon
    si (distAB==distBC) alors
      si (pyth==0) alors
        print("le triangle ABC est isocèle rectangle en B");
      sinon
        print("le triangle ABC est isocèle en B");
      fsi
    sinon
      si (distAC==distBC) alors
        si (pyth==0) alors
          print("le triangle ABC est isocèle rectangle en C");
        sinon
          print("le triangle ABC est isocèle en C");
        fsi
      sinon
        si (pyth==0) alors
          print("le triangle ABC est rectangle en " + le_sommet);
        sinon
          print("le triangle ABC est quelconque");
        fsi
      fsi
    fsi
  fsi
fsi
erase();
A := point(xA,yA);
legende(A,"A");
```

CORRECTION

```
  B := point(xB,yB);
  legende(B,"B");
  C := point(xC,yC);
  legende(C,"C");
  segment(A,B);
  segment(A,C);
  segment(B,C);
};;
```