

INFO0947: Polylignes, Milestone 1

Groupe 06: Maxime DERAUVET, Luca MATAGNE

Table des matières

1	TAD : Point2D	3
1.1	Signature	3
1.2	Sémantique	3
1.3	Structure	3
1.4	Fonctions et procédures	3
2	TAD : Polyligne	4
2.1	Signature	4
2.2	Sémantique	4
2.3	Implémentation par tableau	5
2.3.1	Structure	5
2.3.2	fonctions et procédures	5
2.3.3	Invariant et spécifications : Length	6
2.3.4	Invariant et spécifications : PolyTranslate	7
2.3.5	Invariant et spécifications : PolyRotate	7
2.4	Implémentation par liste chaînée	8
2.4.1	structure	8
2.4.2	Fonctions et procédures	8
3	Complexité	9
4	Comparaison des implémentations	9

1 TAD : Point2D

1.1 Signature

TYPE : Point2D

UTILISE : Réels

OPERATIONS :

- Create : Réels \times Réels \rightarrow Point2D \mathbf{C}^1
- GetX : *Point2D* \rightarrow Réels \mathbf{O}
- GetY : *Point2d* \rightarrow Réels \mathbf{O}
- EuclDist : *Point2D* \times *Point2d* \rightarrow Réels \mathbf{O}
- Translate : *Point2D* \times *Point2d* \rightarrow *Point2D* \mathbf{T}
- Rotate : *Point2D* \times *Point2d* \times Réels \rightarrow *Point2D* \mathbf{T}

1.2 Sémantique

PRECONDITIONS :

AXIOMES : $\forall X, Y \in Reels$

- GetX(Create(a,b)) = a
- GetY(Create(a,b)) = b
- $EuclDist(X,Y) = \sqrt{(GetX(X) - GetX(Y))^2 + (GetY(X) - GetY(Y))^2}$
- GetX(Translate(U,V)) = GetX(U) + GetX(V)
- GetY(Translate(U,V)) = GetY(U) + GetY(V)
- $GetX(Rotate(U,V,f)) = \cos(f) \times (GetX(U) - GetX(V)) - \sin(f) \times (GetY(U) - GetY(V)) + GetX(V)$
- $GetY(Rotate(U,V,f)) = \sin(f) \times (GetX(U) - GetX(V)) + \cos(f) \times (GetY(U) - GetY(V)) + GetY(V)$

1.3 Structure

Un point est créé sur base du couple de ses coordonnées (x,y).

```
1 struct Point2D{
2     float x;
3     float y;
4 };
```

1.4 Fonctions et procédures

```
1 /*
2  * @pre: /
3  * @post: (get_x(create_Point2D) = x
4  * ^
5  * get_y(create_Point2D) = y)
6  */
7
8 Point2D* CreatePoint2D(float x, float y);
9
10 /*
11  * @pre: A != NULL
12  * @post: A = A0 ^ get_x = x
```

1. Les lettres vertes permettront, durant tout ce rapport, de mettre en évidence les observateurs et les opérations internes.

```

12  */
13  float get_x(Point2D* A);
14  /*
15  * @pre: A != NULL
16  * @post: A = A0 ∧ get_y = y
17  */
18  float get_y(Point2D* A);
19  /*
20  * @pre: A != NULL ∧ B != NULL
21  * @post: A = A0 ∧ B = B0 ∧ EuclDist =  $\sqrt{(X_a - X_b)^2 + (Y_a - Y_b)^2}$ 
22  */
23  unsigned float EuclDist(Point2D* A, Point2D* B);
24  /*
25  * @pre: A != NULL ∧ B != NULL
26  * @post: A = Translate(A,B) ∧ B = B0
27  */
28  void TranslatePoint2D(Point2D* A, Point2D* B);
29  /*
30  * @pre: A != NULL ∧ B != NULL
31  * @post: A = Rotate(A,B),x ∧ B = B0
32  */
33  void RotatePoint2D(Point2D* A, Point2D* B, float x);

```

2 TAD : Polyligne

2.1 Signature

TYPE : Polyligne

UTILISE : Point2D, Réels, Naturels, Boolean

OPERATIONS :

- Create : $Point2D \times Point2D \times Boolean \rightarrow Polyligne$
- Close : $Polyligne \rightarrow Polyligne \text{ T}$
- Open : $Polyligne \rightarrow Polyligne \text{ T}$
- IsOpen : $Polyligne \rightarrow Boolean \text{ O}$
- NbrPoint : $Polyligne \rightarrow Naturels \text{ O}$
- GetPoint : $Polyligne \times Naturels \rightarrow Point2D \text{ O}$
- Length : $Polyligne \times Reels \text{ O}$
- AddPoint : $Polyligne \times Point2D \rightarrow Polyligne \text{ T}$
- SuppPoint : $Polyligne \rightarrow Polyligne \text{ T}$
- PolyTranslate : $Polyligne \times Point2D \rightarrow Polyligne \text{ T}$
- PolyRotate : $Polyligne \times Reels \times Point2D \rightarrow Polyligne \text{ T}$

2.2 Sémantique

PRECONDITIONS : $\forall P \in Polyligne, \forall A \in Point2D, \forall x \in Naturels, \forall n \in Boolean$

- SuppPoint(P) est défini ssi $2 \leq NbrPoint(P)$
- GetPoint(A,x) est défini ssi $0 \leq x < NbrPoint(P)$
- AddPoint(P,A) est défini ssi $2 \leq NbrPoint(P)$

AXIOMES : $\forall P \in Polyligne, \forall A, B, C \in Point2D, \forall x \in Naturels, \forall n \in Boolean$

- $Open(Create(A,B,n)) = Create(A,B,n)^2$
- $Close(Create(A,B,n)) = AddPoint(Create(A,B,False),C)$

2. Les Polygones seront toujours ouvertes à la création

- $\text{NbrPoint}(\text{Create}(A,B,n)) = 2$
- $\text{NbrPoint}(\text{AddPoint}(P,C, x)) = \text{NbrPoint}(P) + 1$
- $\text{NbrPoint}(\text{SuppPoint}(P, x)) = \text{NbrPoint}(P) - 1$
- $\text{NbrPoint}(\text{Translate}(P, A)) = \text{NbrPoint}(P)$
- $\text{NbrPoint}(\text{Rotate}(P, A)) = \text{NbrPoint}(P)$
- $\text{GetPoint}(\text{Create}(A,B,n), 0) = A$
- $\text{GetPoint}(\text{AddPoint}(P, C), \text{NbrPoint}(P)) = C$
- $\text{GetPoint}(\text{PolyTranslate}(P, C), x) = \text{Translate}(\text{GetPoint}(P, x), C)$
- $\text{GetPoint}(\text{PolyRotate}(P, C), x) = \text{Rotate}(\text{GetPoint}(P, x), C)$
- $\text{Length}(\text{Create}(A,B,n)) = \text{EuclDist}(A,B)$
- $\text{Length}(P) = \sum_{x=0}^{\text{NbrPoint}(P)-1} \text{EuclDist}(\text{GetPoint}(P, x), \text{GetPoint}(P, x+1))$
- $\text{Length}(\text{Close}(P)) = \text{IF}(\text{IsOpen}(P) = \text{False}) : \text{Length}(P=P_0^3) \text{ ELSE} : \text{Length}(P_0) + \text{EuclDist}(\text{GetPoint}(P, \text{NbrPoint}(P_0)), \text{GetPoint}(P, \text{NbrPoint}(P)))$
- $\text{Length}(\text{Open}(P)) = \text{IF}(\text{IsOpen}(P) = \text{True}) : \text{Length}(P=P_0) \text{ ELSE} : \text{Length}(P_0) - \text{EuclDist}(\text{GetPoint}(P, \text{NbrPoint}(P_0)), \text{GetPoint}(P, \text{NbrPoint}(P)))$
- $\text{Length}(\text{AddPoint}(P)) \ \& \ \text{Length}(\text{SuppPoint}(P))$ ⁴

2.3 Implémentation par tableau

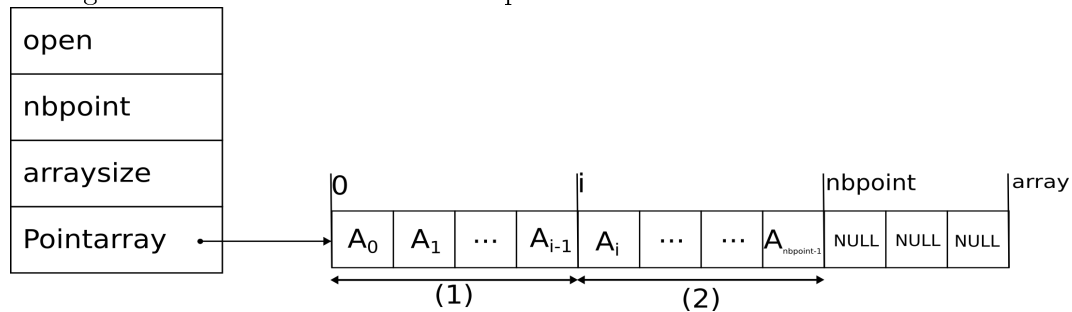
2.3.1 Structure

```

1 struct Polyline{
2     boolean open;
3     unsigned nbpoint;
4     unsigned arraySize;
5     Point2D** pointArray;
6 };

```

L'image ci-dessous est le schéma correspondant à notre structure.



2.3.2 fonctions et procédures

```

1 /*
2  * @pre: A != NULL ^ B != NULL ^
3  * @post: A = A0 ^ B = B0 ^ open=open ^ create_Polyligne = P ^
4  * nbpoint(P) = NbrPoint(P) ^ length(P) = Length(P)
5  */
6 Polyline* CreatePolyline(Point2D* A, Point2D* B, boolean open);
7 /*

```

3. Ici, l'indice "0" est utilisé pour parler de l'état initial de la polyligne, un raisonnement similaire pourrait être utilisé dans le suite du rapport

4. Ces deux cas ne sont pas oubliés mais sont bel et bien pris en compte dans les cas "Length(Open(P))" et "Length(Close(P))" car comme nous le verrons, fermer une polyligne ouverte lui ajoute un point (raisonnement opposé pour l'ouverture d'une polyligne"

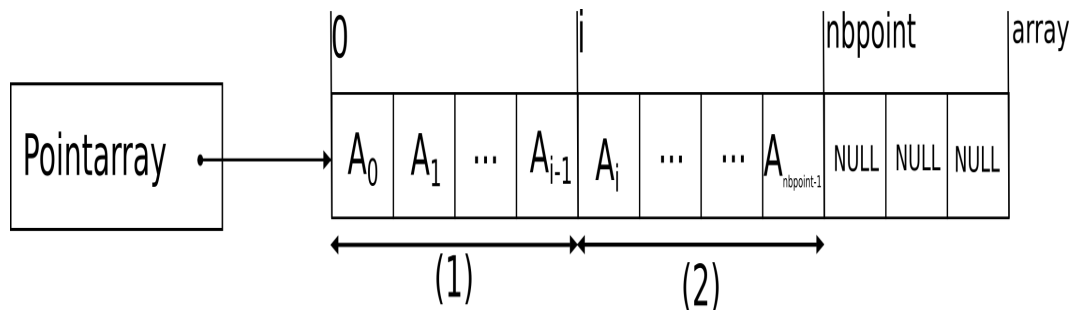
```

8  * @pre: P != NULL
9  * @post: P = P0 ∧ open = False ∧ nbpoint = nbpoint0
10 */
11 void Open(Polyline* P);
12 /*
13  * @pre: P != NULL
14  * @post: P = P0 ∧ open(P) = True ∧ nbpoint = nbpoint0
15  */
16 void Close(Polyline* P);
17 /*
18  * @pre: P != NULL
19  * @post: P = P0 ∧
20  */
21 void IsOpen(Polyline* P);
22 /*
23  * @pre: P != NULL
24  * @post: P = P0 ∧ nbpoint = NbrPoint(P)
25  */
26 unsigned NbrPoint(Polyline* P);
27 /*
28  * @pre: P != NULL ∧ numero < nbpoint
29  * @post: P = P0 ∧ GetPoint = Anumero
30  */
31 Point2D GetPoint(Polyline* P, unsigned numero);
32 /*
33  * @pre: P != NULL ∧ A != NULL
34  * @post: A = A0 ∧ open = open0 ∧ nbpoint = nbpoint0 + 1
35  */
36 void AddPoint2D(Polyline* P, Point2D* A);
37 /*
38  * @pre: P != NULL ∧ A != NULL
39  * @post: A = A0 ∧ open = open0 ∧ nbpoint = nbpoint0 - 1
40  */
41 void SuppPoint2D(Polyline* P);

```

Les fonctions et procédures reprisent ci-dessus ne nécessitent pas d'invariant spécifique. On pourrait imaginer qu'il en faut un pour l'ajout et la suppression d'un point mais nous prenons la décision de n'ajouter et de supprimer que le dernier point à chaque appel de cet fonction. Le seul élément de l'ajout/la suppression d'un point qui pourrait encore nécessiter un invariant est le calcul de la longueur de la polyligne. Une fonction étant totalement dédiée à ce clacul, nous allons présenter l'invariant pour celle-ci.

2.3.3 Invariant et spécifications : Length



(1) : $Length = \sum_{x=0}^{i-2} EuclDist(A_x, A_{x+1})$

(2) : Length à calculer

L'invariant formel qui en découle est le suivant :

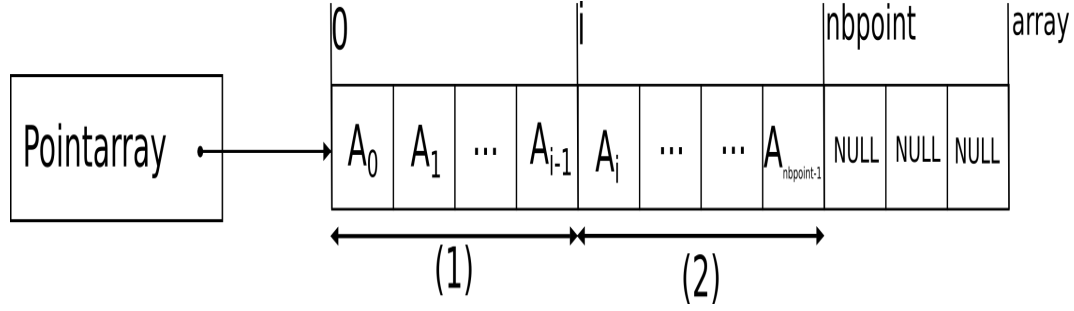
$pointArray = pointArray_0 \wedge \forall x, 0 \leq x \leq i - 2, Length = \sum_{x=0}^{i-2} EuclDist(A_x, A_{x+1}) \wedge$
 $arraySize = arraySize_0 \wedge nbpoint = nbpoint_0$

```

1 /*
2 * @pre: P != NULL
3 * @post: P = P0 ∧ Length =  $\sum_{x=0}^{NbrPoint(P)-1} EuclDist(GetPoint(P, x), GetPoint(P, x+1))$ 
4 */
5 float Length(Polyline* P);

```

2.3.4 Invariant et spécifications : PolyTranslate



(1) : La translation de chaque point est effectuée

(2) : Translation à appliquer sur les points restants

L'invariant formel qui en découle est le suivant :

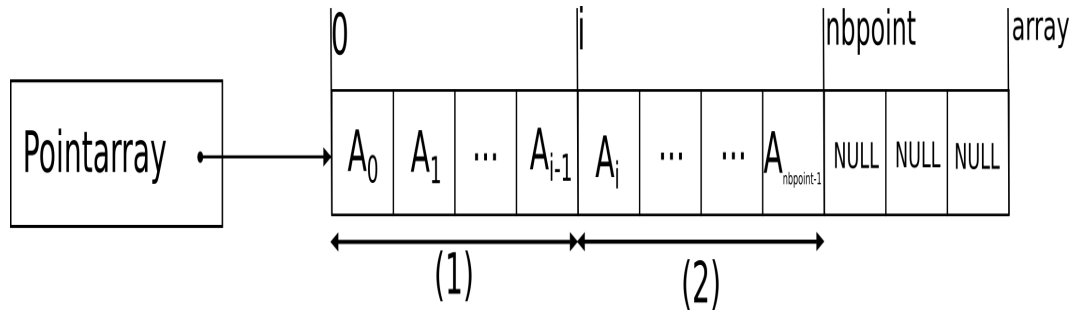
$\forall x, 0 \leq x \leq i - 1, A_x = Translate((A_x)_0, K)^5 \wedge arraySize = arraySize_0 \wedge nbpoint = nbpoint_0$

```

1 /*
2 * @pre: P! = NULL ∧ K! = NULL
3 * @post: P = P0 ∧ C = C0 ∧  $\forall x, 0 \leq x \leq i - 1, A_x = Translate((A_x)_0, K)$ 
4 */
5 void PolyTranslate(Polyline* P, Point2D K);

```

2.3.5 Invariant et spécifications : PolyRotate



(1) : La rotation de chaque point est effectuée

(2) : Rotation à appliquer sur les points restants

L'invariant formel qui en découle est le suivant :

$\forall x, 0 \leq x \leq i - 1, A_x = Rotate((A_x)_0, K)^6 \wedge arraySize = arraySize_0 \wedge nbpoint = nbpoint_0$

```

1 /*
2 * @pre: P! = NULL ∧ K! = NULL
3 * @post: P = P0 ∧ C = C0 ∧  $\forall x, 0 \leq x \leq i - 1, A_x = Rotate((A_x)_0, K)$ 

```

5. K est le point de référence lors de cette Translation

6. K est le point de référence lors de cette Rotation

```

4  */
5  void PolyRotate(ePolyligne* P, Point2D K);

```

2.4 Implémentation par liste chaînée

2.4.1 structure

La structure Point2D est déjà définie plus haut mais il est important de préciser qu'inclure cette structure(Point2D) dans une liste chaînée nécessite de définir une nouvelle structure.

```

1 struct Point{
2     Point* prec;
3     2DPoint* point;
4     Point* suiv;
5 };

```

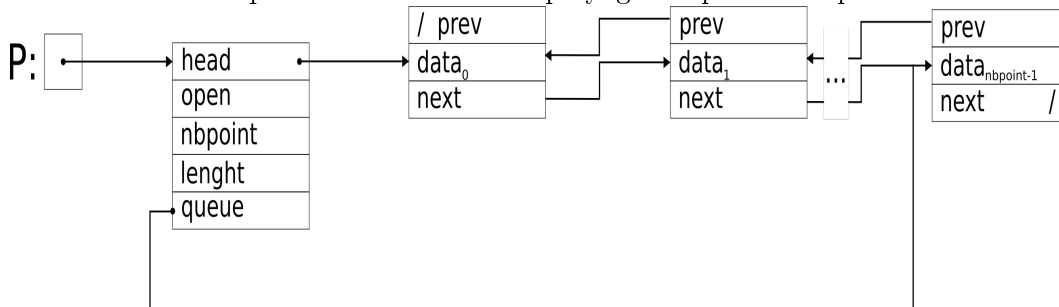
Dans la liste chaînée gérant la polyligne, nous décidons de garder la longueur et le nombre de point dans une cellule d'en-tête afin d'y accéder plus facilement.

```

1 struct Polyline{
2     char open;
3     unsigned nbrpoint;
4     unsigned float length;
5     Point* head;
6 };

```

Voici un schéma représentant le structure polyligne implémentée par une liste chaînée.

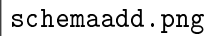


2.4.2 Fonctions et procédures

Les spécifications des fonctions et des procédures sont semblables à l'implémentation par tableau, nous allons juste reprendre deux schémas qui expliquent le fonctionnement des fonctions qui nécessitaient un invariant. Deux schémas et non trois car la longueur de notre polyligne sera stockée dans la cellule d'en tête de notre liste. Il ne sera donc plus nécessaire de parcourir toute la liste afin de la calculer

AddPoint Souvenons nous que notre fonction d'ajout va ajouter notre nouveau point à la fin de la liste. les étapes pour cela seront les suivantes :

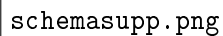
- 1 Amener l'élément courant au dernier élément de la liste
- 2 Initialiser le champ "suiv" de la nouvelle cellule vers la première de la liste
- 3 Initialiser le champ "prec" de la nouvelle cellule vers l'élément courant
- 4 Stocker l'adresse de la nouvelle cellule dans dans le champ "suiv" de l'élément courant
- 5 Stocker l'adresse de la nouvelle cellule dans le champ "prec" du premier élément de la liste



schemaadd.png

SuppPoint Souvenons nous que notre fonction d'ajout va ajouter notre nouveau point à la fin de la liste. les étapes pour cela seront les suivantes :

- 1 Amener l'élément courant au dernier élément de la liste
- 2 Changer le champ "suiv" de l'avant dernier élément pour le champ "suiv" de l'élément courant
- 3 Initialiser le champ "prec" du premier élément pour le champ "prec" de l'élément courant
- 4 supprimer l'élément courant



schemasupp.png

3 Complexité

4 Comparaison des implémentations