Université USTHB – Bab-Ezzouar Bab-Ezzouar, 27 Novembre 2012

Faculté de l’Electronique et de l’Informatique, Département de l’Informatique Année universitaire 2012/2013

Master Informatique, 1ère année, Semestre 1 Semestre 1

Module : Conception et Complexité des Algorithmes

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Corrigé de l'Examen de Contrôle**

**Remarque : Le corrigé proposé ici est détaillé. Lors de la correction des copies, on n'a pas tenu compte de tous les détails qui sont dans ce corrigé. Ces détails sont donnés à titre de rappel et de révision.**

On considère un polynôme P(x) d'une variable réelle x, de degré N (N entier positif ou nul) et de paramètres réels ai, i = 0,…, N (N+1paramètres):

(1)

**1- Algorithmes d'évaluation du polynôme P**

**1.a- Algorithme 1 avec une complexité spatiale O(N) appelé Eval\_1**

Cet algorithme d'évaluation d'un polynôme, appelé **algorithme trivial, naïf ou direct**, effectue un calcul de chaque monôme ai \* xi pour tout i = 0, … , N, et fait le cumul de ces monômes au fur et à mesure. On utilise un tableau noté Tab de (N+1) éléments pour stocker les (N+1) paramètres ai (i=0,…,N) du polynôme P (voir figure 1):

Algorithme Eval\_1 ; //Eval\_1 ≡ Algorithme d'Evaluation 1 avec complexité spatiale O(N)

Var N, i, j : entier ;

Tab : [N+1] reel ; //Tab[i] = ai , i = 0, … , N

x, y, P : réel ;

Debut

//Partie 1: Entrée des données

1 Ecrire('Donner une valeur de N : ') ;

2 Lire(N) ;

3 Ecrire('Donner une valeur de x : ') ;

4 Lire(x) ;

5 i := 0 ; //:= désigne l'instruction d'affectation

6 Tant que (i<=N) faire

Debut

7 Ecrire('Donner une valeur de ai = Tab[i] = ') ;

8 Lire(Tab[i]) ;

9 i := i + 1 ;

Fin ; // Fin Tant que

//Partie 2: Traitement

10 i := 0 ;

11 P := 0 ;

12 Tant que (i<=N) faire

Debut

13 j := 1 ;

14 y := 1 ;

15 Tant que (j<=i) faire //Cette boucle permet de calculer xi, i=0,…,N

Debut

16 y := y \* x ; //y = 1\*x\*…\*x = xi, i=0,…,N

17 j := j + 1 ;

Fin Tant que ; //fin du 2ème Tant que

18 P := P + Tab[i] \* y ; //P = 0 + Tab[0] + Tab[1]\*x1 + …+ Tab[i]\*xi , i=0,…,N

//P = 0 + a0 + a1\*x1 + … + ai \*xi , i=0,…,N

19 i := i + 1 ;

Fin Tant que ; //fin du 1er Tant que

//Partie 3: Sortie des résultats

20 Ecrire('La valeur du polynôme P au point x = ', x, ' est : ', P) ;

Fin. //fin de l'algorithme Eval\_1

**Figure 1.** **Algorithme d'évaluation d'un polynôme P**

**avec une complexité spatiale O(N) (appelé Eval\_1)**

**1.a.1- Calcul de la complexité spatiale de l'algorithme Eval\_1**

**Rappel :**

La complexité spatiale d'un algorithme est la mesure exacte (rarement utilisée) ou en ordre de grandeur (couramment utilisée et donnée en notation Landau) de l'espace mémoire occupé par les instructions et les données lors de son exécution.

**Remarques :**

1- La complexité spatiale est calculée en fonction des données en entrée de l'algorithme.

2- Par ordre de grandeur, il faut comprendre la classe de complexité O(N), O(N2), O(N3), O(log(N)), O(Nlog(N)), O(aN), etc.

3- Généralement, le calcul de la complexité spatiale des algorithmes itératifs ne pose pas de problèmes.

4- Avec les algorithmes récursifs, on doit tenir compte de chaque appel récursif d'une fonction qui génère de façon dynamique et donc de façon transparente au programmeur l'allocation d'un espace mémoire pour sauvegarder les paramètres d'appels de la fonction et des variables locales de la fonction. Dans la pratique, on calcule le nombre d'appels récursifs et on le multiplie par la taille de l'espace mémoire occupé par les paramètres d'appels et les variables locales de la fonction.

5- L'unité de mesure de la complexité spatiale est le mot mémoire qui est un nombre multiple d'octets, mais généralement en puissance de 2 (1, 2, 4 ou 8 octets). On suppose qu'une instruction occupe 1 mot mémoire et une donnée occupe aussi un mot mémoire.

Soit NI le nombre d'instructions de l'algorithme Eval\_1 (ci-dessus) et soit α l'espace mémoire occupé par ces instructions.

On a: NI = 20 instructions.

Comme 1 instruction occupe 1 mot mémoire (hypothèse), alors α = 20 mots mémoires.

Soit ND le nombre de données de l'algorithme Eval\_1 (ci-dessus) et soit β l'espace mémoire occupé par ces données.

On a: 6 variables scalaires N, i, j, x, y et P de type entier et réel; et une variable tableau Tab de (N+1) éléments entiers. Donc: ND = 6 + (N+1) données = N+7 données.

Comme 1 donnée occupe 1 mot mémoire (hypothèse), alors β = N+7 mots mémoires.

Soit S l'espace mémoire occupé par les instructions et les données de l'algorithme Eval\_1.

On a: S = α + β = 20 + (N+7) mots mémoires =.N + 27 mots mémoires.

En conséquence, la complexité spatiale de l'algorithme Eval\_1 est:

* En notation exacte: CS = N+27 mots mémoires.
* En notation Landau: CS = O(N) mots mémoires.

**1.a.2- Calcul de la complexité temporelle de l'algorithme Eval\_1**

**Rappel :**

La complexité temporelle d'un algorithme est la mesure exacte (rarement utilisée) ou en ordre de grandeur (couramment utilisée et donnée en notation Landau) du nombre d'instructions exécutées par cet algorithme.

**Remarques :**

1**-** La complexité temporelle est calculée en fonction des données en entrée de l'algorithme.

2- Par ordre de grandeur, il faut comprendre la classe de complexité O(N), O(N2), O(N3), O(log(N)), O(Nlog(N)), O(aN), etc.

3- Pour calculer la complexité temporelle d'un algorithme, on calcule les fréquences d'exécutions de chacune des instructions de l'algorithme.

4- Généralement, le calcul de la complexité temporelle exacte n'est pas possible, et on se restreint alors au calcul des complexités au pire cas, au moyen cas et au meilleur cas.

5- Avec les algorithmes récursifs, on doit tenir compte de chaque appel récursif d'une fonction qui génère de façon dynamique et donc de façon transparente au programmeur l'exécution de la même fonction avec de nouveaux paramètres d'appels et de nouvelles variables locales de la fonction. Dans la pratique, on calcule le nombre d'appels récursifs en résolvant une équation de récurrence.

6- L'unité de mesure pour la complexité temporelle est le nombre d'instructions exécutées. Les instructions exécutables correspondent aux opérations exécutables par le processeur: les opérations arithmétiques (+, -, \*, /), les opérations de condition (==, !=, <, <=, >, >=) et les opérations d'entrée/sortie (Lire(), Ecrire()). On suppose que toutes ces instructions prennent la même durée de temps d'exécution (ce qui permet de faire l'addition de leurs fréquences d'exécutions).

Soit NI le nombre d'instructions de l'algorithme Eval\_1 (ci-dessus) et soit NX le nombre d'exécutions de toutes les instructions de l'algorithme.

On a: NI = 20 instructions.

Pour calculer NX, on utilise les fréquences d'exécutions de chacune des NI=20 instructions de l'algorithme Eval\_1 comme sur le tableau suivant (voir figure 2):

**N° Instruction Fréquence d'exécution**

//Partie 1: Entrée des données

1 Ecrire('Donner une valeur de N : ') ; 1

2 Lire(N) ; 1

3 Ecrire('Donner une valeur de x : ') ; 1

4 Lire(x) ; 1

5 i := 0 ; 1

6 Tant que (i<=N) faire (N+1)

Debut

7 Ecrire('Donner une valeur

de ai = Tab[i] = ') ; N

8 Lire(Tab[i]) ; N

9 i := i + 1 ; 2N //2 opérations: + et :=

Fin Tant que ;

//Partie 2: Entrée traitement

10 i := 0 ; 1

11 P := 0 ; 1

12 Tant que (i<=N) faire (N+1)

Debut

13 j := 1 ; N

14 y := 1 ; N

15 Tant que (j<=i) faire

Debut

16 y := y \* x ;

17 j := j + 1 ;

Fin Tant que ;

18 P := P + Tab[i] \* y ; (3\*N)

19 i := i + 1 ; (2\*N)

Fin Tant que ;

//Partie 3: Sortie des résultats

20 Ecrire('La valeur du polynôme P

au point x = ', x, ' est : ', P) ; 1

Fin. //fin de l'algorithme Eval\_1

**Figure 2. Tableau des fréquences d'exécutions des instructions de l'algorithme Eval\_1**

On a alors:

En conséquence, la complexité temporelle de l'algorithme Eval\_1 est:

* En notation exacte: instructions.
* En notation Landau: instructions.

**1.b- Algorithme 2 avec une complexité spatiale O(1) appelé Eval\_2**

L'idée de cet algorithme est de reprendre l'algorithme Eval\_1 et au lieu d'utiliser un tableau de (N+1) éléments pour stocker les (N+1) paramètres entiers ai (i=0,…,N) du polynôme P, on utilise une seule (1) variable de type entier notée var1 pour stocker au fur et mesure ces (N+1) paramètres. A chaque lecture d'un élément ai dans la variable var1, le contenu précédent de var1 est écrasé (et bien entendu) perdu.

Ce changement dans la structure de données de l'algorithme nécessite d'imbriquer la partie lecture des données avec la partie traitement. Mais, au niveau traitement, rien ne diffère entre les 2 algorithmes. L'algorithme Eval\_2 est comme suit (voir figure 3):

Algorithme Eval\_2 ; //Eval\_2 ≡ Algorithme d'Evaluation 2 avec complexité spatiale O(1)

Var N, i, j : entier ;

var1 : entier ; //var1 reçoit les paramètres ai , i = 0, … , N

x, y, P : réel ;

Debut

//Partie 1: Entrée des données et traitement

* Ecrire('Donner une valeur de N : ') ;
* Lire(N) ;
* Ecrire('Donner une valeur de x : ') ;
* Lire(x) ;

5 i := 0 ; //:= désigne l'instruction d'affectation

6 P := 0 ;

7 Tant que (i<=N) faire

Debut

8 Ecrire('Donner une valeur de ai = var1 = ') ;

9 Lire(var1) ;

10 j := 1 ;

11 y := 1 ;

12 Tant que (j<=i) faire //Cette boucle permet de calculer xi, i=0,…,N

Debut

13 y := y \* x ; //y = 1\*x\*…\*x = xi, i=0,…,N

14 j := j + 1 ;

Fin Tant que ; //fin du 2ème Tant que

15 P := P + var1\* y ; //P = 0 + a0 + a1\*x1 + … + ai \*xi , i=0,…,N

16 i := i + 1 ;

Fin Tant que ; //fin du 1er Tant que

//Partie 2: Sortie des résultats

17 Ecrire('La valeur du polynôme P au point x = ', x, ' est : ', P) ;

Fin. //fin de l'algorithme Eval\_2

**Figure 3.** **Algorithme d'évaluation d'un polynôme P**

**avec une complexité spatiale O(1) (appelé Eval\_2)**

**1.b.1- Calcul de la complexité spatiale de l'algorithme Eval\_2**

Soit NI le nombre d'instructions de l'algorithme Eval\_2 (ci-dessus) et soit α l'espace mémoire occupé par ces instructions.

On a: NI = 15 instructions.

Comme 1 instruction occupe 1 mot mémoire (hypothèse), alors α = 15 mots mémoires.

Soit ND le nombre de données de l'algorithme Eval\_1 (ci-dessus) et soit β l'espace mémoire occupé par ces données.

On a: 7 variables scalaires N, i, j, var1, x, y et P de type entier et réel. Donc: ND = 7 données.

Comme 1 donnée occupe 1 mot mémoire (hypothèse), alors β = 7 mots mémoires.

Soit S l'espace mémoire occupé par les instructions et les données de l'algorithme Eval\_2.

On a: S = α + β = 15 + 7 mots mémoires =.22 mots mémoires.

En conséquence, la complexité spatiale de l'algorithme Eval\_2 est:

* En notation exacte: CS = 22 mots mémoires.
* En notation Landau: CS = O(1) mots mémoires.

**1.b.2- Calcul de la complexité temporelle de l'algorithme Eval\_2**

De manière similaire au calcul de la complexité temporelle de l'algorithme Eval\_1 fait en 1.a.2, on trouve la complexité temporelle de l'algorithme Eval\_2:

* En notation exacte: instructions.
* En notation Landau: instructions.

**Conclusion :** Les 2 algorithmes Eval\_1 et Eval\_2 ont la même complexité temporelle O(N2) ou, autrement dit, appartiennent à la même classe de complexité temporelle O(N2); mais ils ont des complexités spatiales différentes, respectivement O(N) et O(1).

**2- Algorithme de Horner d'évaluation d'un polynôme P**

Le polynôme P sous la forme (1) peut être réécrit sous la forme (2) équivalente suivante:

(1)

(2)

Pour obtenir la forme (2), on utilise la transformation suivante appelée **algorithme de Horner**:

bN = aN

bi = ai + (bi+1)\*x i = N-1, N-2, … , 2, 1, 0

On a alors: P(x) = b0.

Par exemple, on peut vérifier cette transformation pour N=2:

**2.a- Algorithme de Horner itératif d'évaluation d'un polynôme appelé Eval\_3**

L'écriture de cet algorithme est une traduction directe de la transformation de Horner (voir figure 4):

Algorithme Eval\_3 ; //Eval\_3 ≡ Algorithme d'Evaluation 3

Var N, i : entier ;

Tab : [N+1] reel ; //Tab[i] = ai , i = 0, … , N

x, z, P : réel ;

Debut

//Partie 1: Entrée des données

1 Ecrire('Donner une valeur de N : ') ;

2 Lire(N) ;

3 Ecrire('Donner une valeur de x : ') ;

4 Lire(x) ;

5 i := 0 ; //:= désigne l'instruction d'affectation

6 Tant que (i<=N) faire

Debut

7 Ecrire('Donner une valeur de ai = Tab[i] = ') ;

8 Lire(Tab[i]) ;

9 i := i + 1 ;

Fin ; // Fin Tant que

//Partie 2: Traitement

10 i := N ;

11 z := Tab[i] ;

12 Tant que (i>0) faire

Debut

13 z := Tab[i-1] + (z)\*x ; //Cette instruction contient 4 opérations (dont (i-1))

14 i := i - 1 ; // == 2 ==

Fin Tant que ; //fin du Tant que

15 P := z ;

//Partie 3: Sortie des résultats

16 Ecrire('La valeur du polynôme P au point x = ', x, ' est : ', P) ;

Fin. //fin de l'algorithme Eval\_3

**Figure 4.** **Algorithme d'évaluation de Horner itératif d'un polynôme P**

**avec une complexité spatiale O(N) (appelé Eval\_3)**

**2.a.1- Calcul de la complexité spatiale de l'algorithme Eval\_3**

De manière similaire au calcul de la complexité spatiale de l'algorithme Eval\_1 fait en 1.a.1, on trouve la complexité spatiale de l'algorithme Eval\_3:

NI = 16 instructions et α = 16 mots mémoires.

ND = N + 6 données et alors β = N+6 mots mémoires.

S = α + β = N + 22 mots mémoires.

En conséquence, la complexité spatiale de l'algorithme Eval\_3 est:

* En notation exacte: CS = N+22 mots mémoires.
* En notation Landau: CS = O(N) mots mémoires.

**2.a.2- Calcul de la complexité temporelle de l'algorithme Eval\_3**

De manière similaire au calcul de la complexité temporelle de l'algorithme Eval\_1 fait en 1.a.2, on trouve la complexité temporelle de l'algorithme Eval\_3:

* En notation exacte: instructions.
* En notation Landau: instructions.

**Conclusion :**

On voit bien que l'algorithme de Horner de complexité temporelle O(N) est meilleur que l'algorithme Eval\_1 de complexité temporelle O(N2).

**2.b- Algorithme de Horner récursif d'évaluation d'un polynôme appelé Eval\_4**

Pour obtenir un algorithme de Horner récursif, on utilise la transformation de Horner au polynôme (1) en mettant x en facteur. On obtient alors la forme équivalente (3) ou (4):

(1)

(3)

(4)

On voit de la forme (3) ou (4) que le calcul de P(x) se ramène au calcul d'un sous-polynôme P avec (N-1) paramètres (a1 à aN) et avec le degré de x diminué aussi de 1 (xi-1). De même, ce dernier sous-polynôme peut aussi se ramener au calcul d'un sous-sous-polynôme avec (N-2) paramètres (a2 à aN) et avec le degré de x diminué aussi de 1 (xi-2), etc. La forme (3) ou (4) est donc une forme récursive. L'algorithme est le suivant (voir figure 5):

Algorithme Eval\_4 ; //Eval\_4 ≡ Algorithme d'Evaluation 4

Var N, i : entier ;

Tab : [N+1] reel ; //Tab[i] = ai , i = 0, … , N

x, z, P : réel ;

Debut

//Partie 1: Entrée des données

1 Ecrire('Donner une valeur de N : ') ;

2 Lire(N) ;

3 Ecrire('Donner une valeur de x : ') ;

4 Lire(x) ;

5 i := 0 ; //:= désigne l'instruction d'affectation

6 Tant que (i<=N) faire

Debut

7 Ecrire('Donner une valeur de ai = Tab[i] = ') ;

8 Lire(Tab[i]) ;

9 i := i + 1 ;

Fin ; // fin Tant que

//Partie 2: Traitement

10 z := Horner\_Rec(Tab, N, x) ; //Appel de la fonction Horner\_Rec (Horner récursive)

11 P := z ;

//Partie 3: Sortie des résultats

12 Ecrire('La valeur du polynôme P au point x = ', x, ' est : ', P) ;

Fin. //fin de l'algorithme Eval\_4

Fonction Horner\_Rec(Tab: tableau[0 .. N], N: entier, x: réel): réel;

Var i : entier;

H, p1 : réel;

Debut

1 Si (N==1)

2 Alors Retourner (Tab[0]) ;

Sinon Debut

3 p1 := Tab[0] ; //p1 = premier élément du tableau Tab

4 Pour i := 1 à (N-1) faire

Debut

5 Tab[i-1] := Tab[i] ;

Fin Pour ; //fin Pour

//Cette boucle permet à chaque fois de diminuer le tableau Tab de son //premier élément en faisant un décalage à gauche jusqu'à obtenir un //tableau à 1 élément.

//(Exp: Tab=[1, 2, 3] Tab=[2, 3] (Tab=[3])

6 H := p1 + x \* Horner\_Rec(Tab, N-1, x) ;

//Appel récursif de Horner\_Rec

7 Retourner H;

Fin Si ; //fin Si

Fin ; // fin de la fonction Horner\_rec

**Figure 5.** **Algorithme d'évaluation de Horner récursif d'un polynôme P**

**avec une complexité spatiale O(N) (appelé Eval\_4)**

**2.b.1- Calcul de la complexité spatiale de l'algorithme Eval\_4**

De manière similaire au calcul de la complexité spatiale de l'algorithme Eval\_1 fait en 1.a.1 mais en tenant compte cette fois de chaque appel récursif de la fonction Horner\_Rec qui génère de façon dynamique et donc de façon transparente au programmeur l'allocation d'un espace mémoire pour sauvegarder les paramètres d'appels les variables locales (voir 1.a.1, Remarque 4).

On remarque que le nombre d'appels récursifs est égal N car N est diminué à chaque appel de 1. On multiplie donc N par la taille de l'espace mémoire occupé par les paramètres d'appels et les variables locales de la fonction Horner\_Rec.

On a pour l'algorithme principal de Eval\_4:

NI1 = 12 instructions et α1 = 12 mots mémoires.

ND1 = N + 6 données et β1 = N+6 mots mémoires.

S1 = α1 + β1 = 12 + N + 6 = N + 18 mots mémoires.

Et on a pour la fonction Horner\_Rec:

NI2= 7 instructions et α2 = 7 mots mémoires.

ND2 = N(N + 6) données et β2 = N(N+6) mots mémoires (on a multiplié par N).

S2 = α2 + β2 = 7 + N (N + 6) = N2 + 6N + 7 mots mémoires.

Enfin, pour l'algorithme entier Eval\_4, on a:

S = S1 + S2 = (N+18) + (N2 + 6N + 7) = N2 + 7N + 25 mots mémoires. .

En conséquence, la complexité spatiale de l'algorithme Eval\_4 est:

* En notation exacte: CS = N2 + 7N + 25 mots mémoires.
* En notation Landau: CS = O(N2) mots mémoires.

***2.a.2- Calcul de la complexité temporelle de l'algorithme Eval\_4***

De manière similaire au calcul de la complexité temporelle de l'algorithme Eval\_1 fait en 1.a.2, mais en tenant compte de chaque appel récursif de la fonction Horner\_Rec qui génère de façon dynamique et donc de façon transparente au programmeur l'exécution de la même fonction avec de nouveaux paramètres d'appels et de nouvelles variables locales.

On a pour l'algorithme principal de Eval\_4:

CT = 5N + 10 + H(N) (5)

Et on a pour la fonction Horner\_Rec l'équation de récurrence suivante:

CT2 = H(N) = H(N-1) + 3N + 6 pour tout N>1 et H(0) = 0,

H(N) - H(N-1) = 3N + 6 (6)

Pour (N+1) (6) devient: H(N+1) - H(N) = 3(N+1) + 6 = 3N + 9 (7)

(7) – (6) H(N+1) - 2H(N) + H(N-1) = 3 (8)

Pour (N+1) (8) devient: H(N+2) - 2H(N+1) + H(N) = 3 (9)

(9) – (8) H(N+2) - 3H(N) + 3H(N-1) - 3H(N) = 0 (10)

(10) est une équation de récurrence homogène (car le membre à droite = 0) dont la solution en fonction de N est:

(11)

On remplace dans (5):

(12)

En conséquence, la complexité temporelle de l'algorithme Eval\_4 est:

* En notation exacte: instructions.
* En notation Landau: instructions.

**Conclusion :** Les 2 algorithmes Eval\_3 et Eval\_4 ont la même complexité temporelle O(N2) ou, autrement dit, appartiennent à la même classe de complexité temporelle O(N2); mais ils ont des complexités spatiales différentes, respectivement O(N) et O(N2).

* En notation exacte: instructions.
* En notation Landau: instructions.

**3- Algorithme d'addition de 2 polynômes**

Il suffit d'utiliser l'un des 4 algorithmes d'évaluation d'un polynôme Eval\_1, Eval\_2, Eval\_3 ou Eval\_4 comme une fonction pour évaluer chacun des 2 polynômes P1(x) et P2(x) et ensuite faire l'addition:

P3(x) = P1(x) + P2(x).

En utilisant l'algorithme Eval\_3 de Horner (qui est le plus performant), on a l'algorithme d'addition suivant:

Algorithme Addition ; //addition de 2 polynômes

Var N, i : entier ;

Tab1 : [N+1] reel ; //Tab1[i] = a1i , i = 0, … , N (paramètres du polynôme P1)

Tab2 : [N+1] reel ; //Tab2[i] = a2i , i = 0, … , N (paramètres du polynôme P2)

x : réel ;

Debut

//Partie 1: Traitement

1 P1 := Eval\_3(Tab1,x) ; //appel de la fonction Eval\_3

2 P2 := Eval\_3 (Tab2,x) ; //appel de la fonction Eval\_3

3 P3 := P1 + P2;

//Partie 2: Sortie des résultats

4 Ecrire('La valeur du polynôme P3 au point x = ', x, ' est : ', P3) ;

Fin. //fin de l'algorithme Addition

**4- Algorithme de multiplication de 2 polynômes**

De manière similaire à l'algorithme d'addition et en choisissant d'utiliser l'algorithme Eval\_3, on a l'algorithme de multiplication suivant:

Algorithme Multiplication ; // multiplication de 2 polynômes

Var N, i : entier ;

Tab1 : [N+1] reel ; //Tab1[i] = a1i , i = 0, … , N (paramètres du polynôme P1)

Tab2 : [N+1] reel ; //Tab2[i] = a2i , i = 0, … , N(paramètres du polynôme P2)

x : réel ;

Debut

//Partie 1: Traitement

1 P1 := Eval\_3(Tab1,x) ; //appel de la fonction Eval\_3

2 P2 := Eval\_3 (Tab2,x) ; //appel de la fonction Eval\_3

3 P3 := P1 \* P2;

//Partie 2: Sortie des résultats

4 Ecrire('La valeur du polynôme P3 au point x = ', x, ' est : ', P3) ;

Fin. //fin de l'algorithme Multiplication