**Université Hassan 2 Ecole Normale Supérieure de l'Enseignement Technique Département Mathématique/Informatique (BDCC)**

**Mohammedia**

**Compte-Rendu de TP2 + Application d’integrale**

**Réalisé par : ELYOUSFI Mohamed**

**Classe : II-BDCC1**

**Supervisé par : Pr. M Qbadou**

TP 2 : Polynômes partie 1

**DESCRIPTION :**

L’objet de ce TP est de définir des diverses opérations des polynômes, concernant :

* *La saisie des données d’un polynôme*
* *L’affichage de cette polynôme de la forme : a\*x^0 + b\*x^1+…+n\*x^n*
* *Evaluer la valeur du polynôme étant donnée la valeur de la variable x*
* *Faire l’addition de deux polynômes P et Q fournis en paramètres*
* *Changer le signe d’un polynôme en changeant le signe de tous ses coefficients*
* *Faire la soustraction de deux polynômes*
* *Faire la comparaison de deux polynômes*
* *Calculer la dérivée d’un polynôme*

**ANALYSE FONCTIONNELLE :**

Premièrement on définit une structure de données qui va représenter le polynôme, qui contient un entier **n** qu’est la dégrée de polynôme (degree\_max défini est 1000), et un tableau des réels « **float a[DegM+1]** » qui va stocker les coefficients, les coefficients sont de a**0** à a**n** c’est pour ça le tableau déclaré de **DegM+1.**

Ensuite il ya la fonction « **polynome** **saisirPoly ()** » : qui retourne le polynôme saisi par l’utilisateur.

**« polynome sommePoly (polynome p1,polynome p2) »**: calcul la somme de p1 et p2 et retourne la polynome somme, cette polynome et de degré NMax(p1.n,p2.n)

**« polynome soustractionPoly(polynome p1,polynome p2) » :** fait la soustraction de p1 et p2

Et retourne la polynome résultante.

**« void changeSigne(polynome \*poly) » :** change le signe d’une polynome par le changement de signes de ses coeficients.

**«void comparePoly(polynome p1,polynome p2) »:** affiche si les polynômes p1 et p2 sont égaux ou pas.

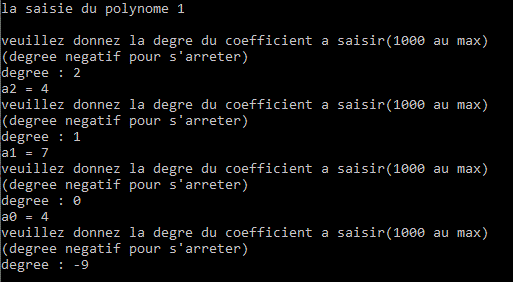
**« polynome derivPoly(polynome poly) » :** calcul et retourne la dérivée d’une polynome

**« void affichePoly(polynome poly) » :** affiche tout simplement la polynome en paramètre

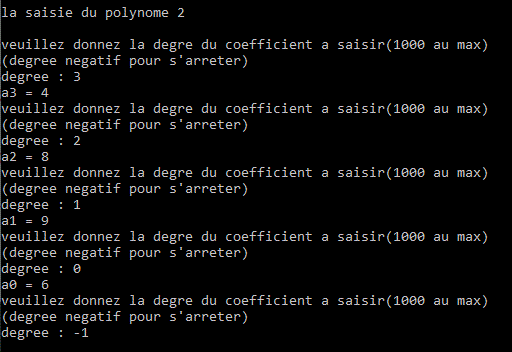
**«float evaluer(polynome poly, float x) » :** évaluer la valeur poly(x) et la retourner

**EXECUTION :**

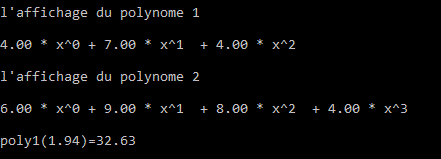
La saisie de la première polynome **4X2+7X+4**, il s’agit d’entrer la dégrées de chaque coeficients avant le saisir, pour arrêter il faut entrer une dégrée négatif

****

La saisie de la deuxième polynome **4X3+8X2+9X+6** de la même maniere :



Affichage des deux polynômes saisie **4X2+7X+4** et **4X3+8X2+9X+6**

**

Les différents opérations des polynômes :

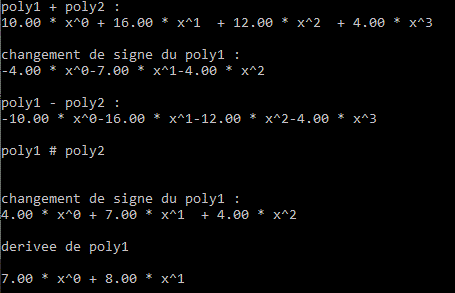
**La somme** (4X2+7X+4) + (4X3+8X2+9X+6) = 4X3 + 12X2 + 16X + 10

**Le changement de signe de p1** (-4X2-7X-4)

**La soustraction** (-4X2-7x-4) - (4X3+8X2+9x+6) = -4X3 - 12X2 – 16X - 10

**La comparaison**

**La dérivée**

**

TP 2 : Polynômes partie 2

**DESCRIPTION :**

L’idée c’est qu’on souhaite implémenter une solution qui calcule une racine approché d’un polynôme en partant d’une 1ère valeur choisie aléatoirement.

*Il existe plusieurs méthodes de calcul numérique permettant d’approcher les valeurs des racines comme La méthode de Newton-Raphson qu’on va implémenter dans ce programme.*

*Cette méthode consiste à prendre une première estimation (aléatoirement ou non) d’une valeur de x que l’on espère proche d’une racine, puis à raffiner cette valeur en suivant la pente de la fonction donnée par sa dérivée.*

*Formellement, il s’agit de calculer la suite par la forme itérative suivante :*

**

*Où r est l’estimation initiale de la racine et xk l’estimation raffinée de cette racine. Plus la valeur de k est élevée,plus l’approximation sera précise*

*.*

**ANALYSE FONCTIONNELLE :**

Premièrement on définit une structure de données qui va représenter le polynôme, qui contient un entier **n** qu’est la dégrée de polynôme (degree\_max défini est 1000), et un tableau des réels « **float a[DegM+1]** » qui va stocker les coefficients, les coefficients sont de a**0** à a**n** c’est pour ça le tableau déclaré de **DegM+1.**

Ensuite il ya la fonction « **polynome** **saisirPoly ()** » : qui retourne le polynôme saisi par l’utilisateur.

**« polynome derivPoly(polynome poly) » :** calcul et retourne la dérivée d’une polynome

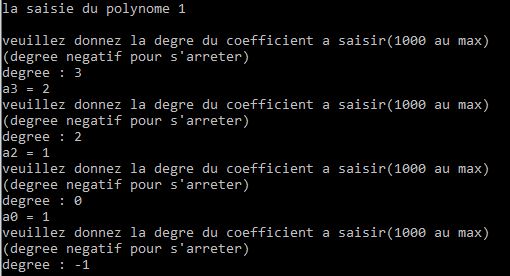
**« void affichePoly(polynome poly) » :** affiche tout simplement la polynome en paramètre

**« float evaluer (polynome poly, float x) » :** évaluer la valeur poly(x) et la retourner

**«*void calculRacine (polynome poly, polynome drvPoly, double x0, int k)* » :** qui calcul et affiche la racine par la méthode de Newton-Raphson.

**EXECUTION**

La saisie de la polynome **2X3+1X2+1**



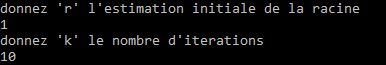
L’affichage cette polynome

****

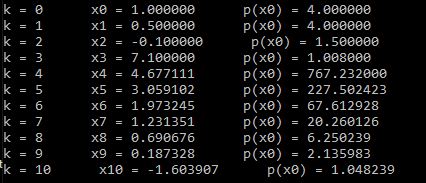
La dérivée de 2X3+1X2+1 **=** **6X2 + 2X**



La saisi de l’estimation r (expliqué dans la description), et k le nombre d’itérations



Maintenant on a arrivé au but du programme qu’est le calcul d’une racine approchée

**

Applications d’intégrale

**DESCRIPTION :**

Il s’agit de calculer l’integrale des fonctions scalaire (exp(x), X2, sqrt(x)) dans un intervalle [a,b] a saisi par l’utilisateur.

Ce programme peut être fait par plusieurs méthodes j’ai choisi de travailler avec la méthode des trapèzes en vue de son efficacité

Cette application permet de familiariser avec les pointeurs sur le fonctions.

**ANALYSE FONCTIONNELLE :**

Les fonctions utilisées :

* **« float F\_expX(float x) »** : c’est une fonction qui calcul et retourne l’évaluation d’exponentielle d’un réel x donnée comme paramètre
* **« float F\_carreX(float x) » :** c’est une fonction qui calcul et retourne l’évaluation du carrée d’un réel x donnée comme paramètre
* **« float F\_sqrtX(float x) » :** c’est une fonction qui calcul et retourne l’évaluation du racine carrée d’un réel x donnée comme paramètre (x dans ce cas doit être >= 0)
* **« float integrale\_F(float (\*p)(float) » :** c’est une fonction qui calcul et retourne l’intégrale d’une fonction donnée comme paramètre, à l’aide d’un pointeur sur le fonctions ayant comme type de retour float et comme paramètre un nombre réel (float).

***EXECUTION :***

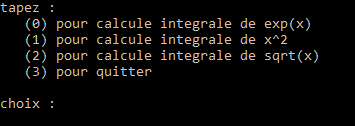
Au départ le programme affiche un menu

0 pour calculer l’intégrale d’exponentielle(x) dans un intervalle [a,b] donnée

1 pour calculer l’intégrale de X2

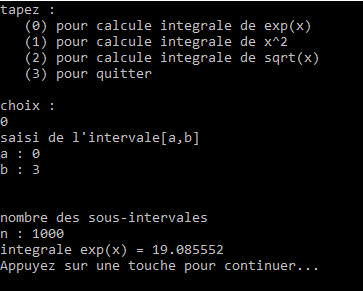
2 pour calculer l’intégrale de √X

Et finalement 3 pour quitter



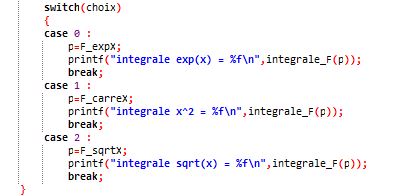
Pour l’exponentielle :

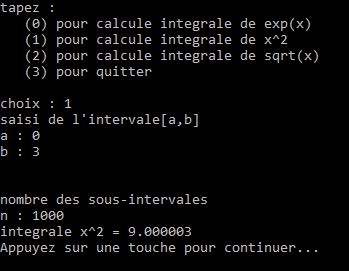
Just après le choix, le programme demande à l’utilisateur de saisir l’intervalle [a, b] puis d’entrer le nombre des sous-intervalles n, il s’agit de combien on va décomposer notre intervalle [a, b] plus n est grande plus la valeur est précise, et ensuite le programme affiche le résultat.

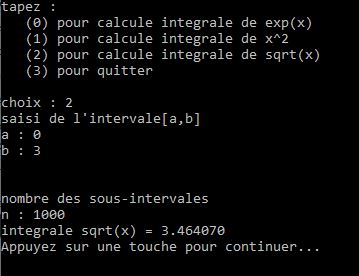
**

J’ai utilisé un pointeur qui pointe vers des fonctions de signature **float nom(float)**

A chaque choix d’utilisateur, le pointeur pointe vers une différente fonction.

**

C’est la même chose pour les deux autres fonctions



**La formule utilisé dans les calculs**

