Rapport de projet LO21

Dans le cadre de l'unité de valeur LO21 du Tronc Commun de l'UTBM, nous avons été amenés à travailler sur un projet informatique en relation directe avec les mathématiques. En effet, le but de ce projet était de fournir une bibliothèque en langage C pour permettre la manipulation de polynômes. Ce rapport vient donc comme un support pour comprendre le code commenté.

I. Présentation du travail demandé

Le produit final du projet doit être une bibliothèque pour une manipulation basique des polynômes, ainsi qu'un exécutable permettant à un éventuel utilisateur de pouvoir utiliser simplement cet ensemble de fonctions. Tout ceci implémenté en langage C, réutilisant ainsi les notions assimilées tout au long de l'unité de valeur.

Afin de modéliser informatiquement un polynôme, plusieurs solutions et structures sont possibles, mais l'énoncé ne nous laissait pas libre sur ce point : nous devons utiliser une liste doublement chaînée pour représenter un polynôme. La définition d'un nouveau type en tant que structure pour la liste doublement chaînée était également imposée. Les champs des éléments composant la liste chaînée nous a aussi été communiquée dans l'énoncé du travail. Par ailleurs, un polynôme étant composé de différents monômes, il nous fallu également définir un nouveau type de type structure pour modéliser les monômes. Toutes ces définitions peuvent être retrouvées dans le header polynome.h.

Cette première étape de modélisation informatique des éléments est essentielle pour la bonne continuation du projet. Cette étape a en effet été préparée par le professeur ayant fait le sujet, mais on peut aussi imaginer comment pourrait se présenter le problème si on décidait de changer les structures utilisées.

La seconde étape du projet est donc le codage des sous-programmes, des fonctions pour la manipulation des polynômes. On nous fournit une liste minimale de fonction devant apparaitre dans le projet, nous laissant en même temps la liberté d'en ajouter pour simplifier la programmation ou pour ajouter d'éventuelles fonctionnalités. Les prototypes de ces fonctions sont rassemblés dans le header polynome.h, alors que la définition de chaque fonction apparait dans le fichier source polynome.c, comme nous l'indique encore une fois l'énoncé.

La dernière étape, après que tout ceci soit codé, est la réalisation du programme principal qui permettra à n'importe quel utilisateur, possédant une connaissance minimale sur les polynômes, de pouvoir utiliser les fonctions pour satisfaire ses besoins.

II. Description des structures de données

A. Les monômes

```
typedef struct
{
     float coef;
     int deg;
} Monome;
```

Les monômes sont définis comme une structure avec 2 champs : un champ pour le coefficient devant la variable du monôme, un champ pour l'ordre de multiplicité, le degré, de la variable du monôme. On généralise cette structure en définissant un nouveau type : Monome.

B. Les polynômes et ses éléments

```
typedef struct elem
{
     Monome monome;
     struct elem *next;
     struct elem *prev;
} Element;

typedef struct
{
     Element *tete;
     Element *last;
     int taille;
     int hdeg;
} Polynome;
```

Les polynômes sont définis comme la structure d'une liste doublement chaînée. On peut accéder à son premier et dernier élément, à sa taille et au plus haut degré contenu dans ce polynôme. Chaque élément est lui-même défini comme une structure, défini par les champs suivants : un monôme, un pointeur vers l'élément suivant, un pointeur vers l'élément précédent. C'est en effet ce pointeur vers l'élément précédent qui rend la liste doublement chaînée. On généralise ces structures en créant deux types : Element et Polynome.

On modélise ainsi un polynôme sous la forme d'une liste doublement chaînée, composée d'éléments reliés entre eux de manière réciproque, composés chacun d'un monôme du polynôme.

III. Description des sous-programmes manipulateurs

Nous chercherons dans cette partie à détailler le fonctionnement et les caractéristiques principales de chaque sous-programme.

- **Sous-programmes imposés :**

1. creerMonome : Réel x Entier → Monome

Cette fonction prend en paramètre un réel, qui correspond au coefficient du monôme, ainsi qu'un entier, correspondant au degré du monôme, et renvoie un monôme. Voici le prototype de la fonction :

```
Monome creerMonome (float, int);
```

La fonction déclare simplement un nouveau monôme, puis affecte les valeurs du coefficient et du degré dans les bons champs, puis renvoie le monôme ainsi généré.

2. creerPolynome : Ø → Polynome

Cette fonction ne prend aucun paramètre, et renvoie un polynôme. Voici le prototype de la fonction :

```
Polynome creerPolynome ();
```

La fonction déclare simplement un nouveau polynôme, puis initialise ce dernier comme un polynôme nul, un polynôme vide. L'élément de tête et de queue sont initialisés à NULL, alors que la taille du polynôme ainsi que son plus haut degré sont initialisés à 0. On renvoie ensuite le polynôme ainsi généré.

3. ajouterMonome : Polynome x Monome → Polynome

Cette fonction prend en paramètre un polynôme, correspondant au polynôme qui recevra un nouveau monôme, et un monôme, qui correspond au monôme à ajouter dans le polynôme passé en paramètre, et renvoie un polynôme. Voici le prototype de la fonction :

```
Polynome ajouterMonome (Polynome, Monome);
```

Cette fonction ajoute le monôme au début, au milieu ou à la fin du polynôme, en fonction du degré du monôme. Tout d'abord, on vérifie que le monôme n'est pas « nul », c'est-à-dire que son coefficient n'est pas égal à 0, sinon l'ajout d'un tel monôme revient à renvoyer le polynôme lui-même. Ensuite, on place un pointeur sur le premier élément du polynôme, pointeur qui servira par la suite à parcourir le polynôme. On vérifie aussi que le polynôme n'est pas nul, sinon l'ajout se limitera à un simple ajout en queue ou en tête (on choisi ici d'ajouter arbitrairement en queue).

Une fois ces vérifications passées, on s'intéresse ensuite au monôme du premier élément du polynôme, en le comparant notamment à notre monôme à ajouter. Si les deux possèdent le même degré, on fait la somme des coefficients de ces monômes : si la somme est nulle, on supprime le premier monôme du polynôme, sinon, on met à jour le coefficient du premier monôme du polynôme. Si les deux ne possèdent pas le même degré et que le degré du monôme à ajouter est inférieur au premier degré du premier monôme du polynôme, alors on ajoute notre monôme en tête du polynôme. Si le degré de notre monôme est supérieur au degré du monôme de tête, on entame un parcours du polynôme.

On parcourt le polynôme tant que l'on n'est pas arrivé à la fin de ce dernier et que le degré de notre monôme est supérieur au degré du monôme pointé par notre pointeur créé au départ, on fait avancer ce pointeur d'un élément. Lorsque la boucle s'arrête, cela peut être dû à trois choses : soit le degré de notre monôme est égal au degré du monôme pointé, soit le degré de notre monôme est inférieur au degré du monôme pointé, soit nous sommes arrivés à la fin du polynôme ce qui signifie que notre monôme a un degré supérieur au plus haut degré du polynôme (on aurait d'ailleurs pu vérifier dès le départ si notre monôme possède un degré supérieur au plus haut degré du polynôme). Il reste à identifier dans quel cas on se trouve et à appliquer pour chaque cas le traitement adéquat : si les degrés sont égaux on fait le même traitement que dans le paragraphe ci-dessus, si le degré du monôme est inférieur à celui du monôme pointé on introduit notre monôme entre deux autres en modifiant les pointeurs suivant et précédents, sinon on ajoute en queue notre monôme.

L'ajout en tête et en queue est délégué à des fonctions créées spécialement à cet effet, explicitées plus tard. L'ajout entre deux monômes se fait par un travail sur les pointeurs formant la structure de chaque élément, et en incrémentant la taille du polynôme. On termine par renvoyer le polynôme, intact ou modifié.

4. supprimerMonome : Polynome x Entier → Polynome

Cette fonction prend en paramètre un polynome, qui correspond au polynôme auquel on désire supprimer un monôme, et un entier, correspondant au degré du monôme que l'on veut ôter. On renvoie ensuite un polynôme. Voici le prototype de la fonction :

```
Polynome supprimerMonome (Polynome, int);
```

Cette fonction supprime le monôme de degré entré en paramètre à un polynôme lui aussi entré en paramètre. On vérifie tout d'abord que le polynôme n'est pas nul, sinon la suppression ne peut s'opérer. Ensuite, on place un pointeur sur le premier élément du polynôme, pointeur qui servira par la suite à parcourir le polynôme. Si le degré entré correspond au degré du premier monôme du polynôme, on supprime l'élément de tête du polynôme, sinon on entame le parcours du polynôme.

On parcourt le polynôme tant que l'on n'est pas arrivé à la fin du polynôme et que le degré du monôme pointé est inférieur au degré entré en paramètre. Si l'on sort de la boucle, ce ne peut être que pour trois raisons : soit on est arrivé à la fin du polynôme, soit notre degré est égal à celui du monôme pointé, soit le degré du monôme pointé est supérieur au degré entré en paramètre. Il ne reste plus qu'à identifier dans quel cas on se trouve, et à appliquer le traitement approprié. Si les degrés sont égaux, on supprime le monôme du polynôme associé à ce degré, sinon c'est que le degré que l'on cherche n'est pas dans le polynôme car ceux-ci sont rangés par ordre croissants et que l'on ne doit donc que renvoyer le polynôme tel quel.

La suppression en tête est assurée par une autre fonction créée spécialement à cet effet, explicitée plus tard. Pour supprimer le polynôme associé à un degré on exécute un travaille sur les pointeurs suivants et précédents, en modifiant éventuellement le plus haut degré et le pointeur sur la queue du polynôme, sans oublier de décrémenter la taille de ce dernier. Il ne faut pas omettre de vider la mémoire occupée par cet élément, et de renvoyer le polynôme ainsi modifié.

5. additionner : Polynome x Polynome → Polynome

Cette fonction prend comme paramètre deux polynômes, correspondants aux polynômes à ajouter entre eux, et on renvoie ensuite un polynôme. Voici le prototype de la fonction :

```
Polynome additionner (Polynome, Polynome);
```

Cette fonction prend deux polynômes les additionne terme à terme, et stocke chaque nouveau monôme dans un nouveau polynôme que l'on renvoie à la fin. On vérifie tout d'abord qu'aucun des deux polynômes n'est nul, car il suffirait sinon de renvoyer uniquement le polynôme non nul. On place ensuite un pointeur sur la tête de chaque polynôme, pour les parcourir chacun son tour plus facilement par la suite.

On rentre ensuite dans une boucle, d'où l'on pourra sortir que si l'on est arrivé à la fin d'un des polynômes. On vérifie tout d'abord, à chaque boucle, si les degrés des monômes pointés sont égaux, et dans ce cas on fait la somme des coefficients : si la

somme n'est pas nulle on crée un nouveau monôme de coefficient la somme et de même degré et on l'ajoute à un nouveau polynôme créé. On avance donc d'un élément dans le parcours de chaque polynôme. Si les degrés ne sont pas les mêmes, on regarde quel est le polynôme dont le monôme pointé a le degré le plus petit, puis on crée un nouveau monôme similaire à celui pointé dont le degré est le plus petit, et on l'ajoute à notre polynôme stockant la somme. On avance ensuite d'un élément dans le polynôme concerné. On répète cela jusqu'à ce que l'on sorte de la boucle.

Une fois sortis de la boucle, on regarde quel est le polynôme que l'on a parcouru en entier. On ajoute ensuite tous les termes restants de l'autre polynôme au polynôme somme que l'on va renvoyer juste après.

6. multiplier : Polynome x Polynome → Polynome

Cette fonction prend comme paramètre deux polynômes, correspondants aux polynômes à multiplier entre eux, et on renvoie ensuite un polynôme. Voici le prototype de la fonction :

```
Polynome multiplier (Polynome, Polynome);
```

Cette fonction multiplie deux polynômes en prenant le premier terme du premier polynôme et en le faisant multiplier par chaque terme du second polynôme et en ajoutant à chaque multiplication le monôme formé à un troisième polynôme. On répète tout ceci pour chaque monôme du premier polynôme, et on renvoie enfin le troisième monôme, contenant le produit des deux premiers. On vérifie tout d'abord qu'aucun des deux polynômes n'est nul, car il suffirait sinon de renvoyer le polynôme nul. On place ensuite un pointeur sur la tête de chaque polynôme, pour les parcourir chacun son tour plus facilement par la suite.

On rentre ensuite dans une boucle, d'où l'on ne sortira que lorsque l'on aura parcouru tout le premier polynôme. On rentre ensuite dans une boucle, d'où l'on ne sortira que lorsque l'on aura parcouru tout le second polynôme. A chaque passage de cette dernière boucle on fait le produit des coefficients et la somme des degrés des deux monômes pointés. On crée un monôme de coefficient le produit de coefficients et de degré la somme des degrés, et on l'ajoute au troisième polynôme. On passe ensuite au prochain élément du second polynôme. On refait tout ceci jusqu'à ce que le second polynôme soit totalement parcouru. Une fois sortis de la seconde boucle, on passe à l'élément suivant du premier polynôme et on remet le pointeur du second polynôme à sa tête. On renvoie enfin le troisième polynôme formé.

7. mderiver : Monome x Entier → Monome

Cette fonction prend comme paramètre un monôme, qui correspond au monôme à dériver, un entier, correspondant au nombre de fois que l'on doit dériver le monôme, et on renvoie enfin un monôme. Voici le prototype de la fonction :

```
Monome mderiver (Monome , int);
```

Cette fonction dérive un monôme un certain nombre de fois, et renvoie le monôme ainsi formé. On dérive le monôme tant que celui-ci ne devient pas nul et que l'on n'a pas dérivé le bon nombre de fois le monôme. On renvoie ensuite le monôme ainsi formé. Pour dériver une fois le monôme on utilise une fonction créée spécialement à cet effet, explicitée plus tard dans le rapport.

8. pderiver: Polynome x Entier → Polynome

Cette fonction prend comme paramètre un polynôme, qui correspond au polynôme à dériver, un entier, correspondant au nombre de fois que l'on doit dériver le polynôme, et on renvoie enfin un polynôme. Voici le prototype de la fonction :

```
Polynome pderiver (Polynome, int);
```

Cette fonction dérive un polynôme un certain nombre de fois, et renvoie le polynôme ainsi formé. On dérive le polynôme tant que l'on n'a pas dérivé le bon nombre de fois le polynôme. On renvoie ensuite le polynôme ainsi formé. Pour dériver une fois le polynôme on utilise une fonction créée spécialement à cet effet, explicitée plus tard dans le rapport.

9. afficherMonome : Monome → Ø

Cette procédure prend en paramètre un monôme et l'affiche. Voici le prototype de la procédure :

```
void afficherMonome (Monome);
```

Cette procédure affiche le monôme selon les notations mathématiques habituelles. On doit donc distinguer plusieurs cas : si le degré vaut 0 on n'affiche que le coefficient, sinon si le degré vaut 1 on n'affiche pas le degré au dessus de la variable, sinon si on affiche le monôme avec son coefficient sa variable et le degré. Par ailleurs si le coefficient vaut 1, on n'affiche pas le coefficient avant la variable (sauf si le degré vaut 0).

10. afficherPolynome : Polynome → Ø

Cette procédure prend en paramètre un monôme et l'affiche. Voici le prototype de la procédure :

```
void afficherPolynome (Polynome);
```

Cette procédure affiche un polynôme selon les notations mathématiques usuelles. On vérifie d'abord si le polynôme n'est pas nul, car son affichage sinon est trivial. On affiche donc sinon le polynôme sous la forme d'une somme de ses monômes. Ensuite, on créé un pointeur sur le premier élément du polynôme pour mieux le parcourir. On affiche donc ses monômes avec un signe + juste après sauf le dernier monôme. On utilise pour cela une boucle qui s'arrête jusqu'à ce que l'on pointe sur le dernier élément du polynôme. On affiche ensuite seul le dernier monôme.

- <u>Sous-programmes supplémentaires :</u>

11. ajouterMonomeTete : Polynome x Monome → Polynome

Cette fonction prend en paramètre un polynôme, correspondant au polynôme auquel on va ajouter un monôme en tête, et un monôme, le monôme à ajouter en tête, et on renvoie un polynôme. Voici le prototype de la fonction :

```
Polynome ajouterMonomeTete (Polynome, Monome);
```

Cette fonction ajoute un monôme à la tête d'un polynôme. On ne vérifie pas si le polynôme est nul ou si le monôme est nul car cette fonction n'est utilisée que dans des conditions où l'on est sûr qu'il n'y aura pas de problème. On créé donc un nouvel élément auquel on va attribuer en tant que monôme celui entré en paramètre. On initialise ce nouvel élément en tant que tête de polynôme, modifiant ainsi les champs suivant et précédent de l'élément ainsi que la tête du nouveau polynôme. Nous n'oublions pas de d'incrémenter la taille du polynôme ainsi formé. Selon si le polynôme est nul ou pas, on modifie ou pas le plus haut degré ainsi que le pointeur vers le dernier élément. Il ne reste plus qu'à renvoyer le polynôme ainsi modifié.

12. ajouterMonomeQueue : Polynome x Monome → Polynome

Cette fonction prend en paramètre un polynôme, correspondant au polynôme auquel on va ajouter un monôme en queue, et un monôme, le monôme à ajouter en queue, et on renvoie un polynôme. Voici le prototype de la fonction :

```
Polynome ajouterMonomeQueue (Polynome, Monome);
```

Cette fonction ajoute un monôme à la fin d'un polynôme. On ne vérifie pas si le polynôme est nul ou si le monôme est nul car cette fonction n'est utilisée que dans des

conditions où l'on est sûr qu'il n'y aura pas de problème. On créé donc un nouvel élément auquel on va attribuer en tant que monôme celui entré en paramètre. On initialise ce nouvel élément en tant que queue du polynôme, modifiant ainsi les champs suivant et précédent de l'élément ainsi que la queue du nouveau polynôme. On modifie également le plus haut degré qui devient celui du monôme ajouté, la taille est incrémentée, et selon si le polynôme est nul pour pas, on rattache notre nouveau monôme à l'ancienne queue. Il ne reste plus qu'à renvoyer le monôme ainsi formé.

13. supprimerMonomeTete : Polynome → Polynome

Cette fonction prend en paramètre et renvoie également un polynôme. Voici le prototype de la fonction :

```
Polynome supprimerMonomeTete (Polynome);
```

Cette fonction supprime le premier élément du polynôme passé en paramètre, en accédant directement grâce à un champ à son élément de tête. On ne vérifie pas si le polynôme est nul ou pas car cette fonction n'est utilisée que dans des conditions où l'on est sûr qu'il n'y aura pas de problème. On vérifie par contre si le polynôme est de taille 1 ou pas. S'il l'est, la suppression engendrera un polynôme nul : on se contente de renvoyer un polynôme nul. S'il ne l'est pas, le pointeur sur la tête passera à l'élément suivant qui verra son pointeur précédent passer à NULL. On libère la place précédemment occupée, et décrémente la taille du polynôme, et on renvoie (gentiment) ce dernier.

14. mderiverUnefois : Monome → Monome

Cette fonction prend en paramètre un monôme, correspondant au monôme à dériver une fois, et renvoie un monôme. Voici le prototype de la fonction :

```
Monome mderiverUnefois (Monome);
```

Cette fonction prend un monôme et le dérive une seule fois. On utilise la formule mathématique pour dériver un polynôme, qui nous dit que la dérivée de $k.x^n$ est $k.n.x^{n-1}$. On vérifie d'abord que le monôme n'est pas nul (donc que son coefficient est différent de zéro), sinon il suffirait de le renvoyer, car sa dérivée est égale à lui-même. Une fois ceci vérifié, on applique la formule précédente, et au cas où le degré du monôme ne serait pas nul, on le décrémente. Si le degré est nul, le produit du coefficient et du degré rendra le monôme nul : on dérive bien ce monôme. On renvoie ensuite le monôme ainsi dérivé.

15. pderiverUnefois : Polynome → Polynome

Cette fonction prend en paramètre un polynôme, correspondant au polynôme à dériver une fois, et renvoie un polynôme. Voici le prototype de la fonction :

```
Polynome pderiverUnefois (Polynome);
```

Cette fonction prend un polynôme et le dérive une seule fois. On utilise le fait que la dérivée d'un polynôme est égale à la somme des dérivées de ses monômes. On fixe donc un pointeur sur le premier élément du polynôme pour ensuite parcourir ce dernier. On créé également un second polynôme pour accueillir les nouveaux monômes dérivés. On rentre ensuite dans une boucle qui se termine lorsque l'on a parcouru tout le polynôme. A chaque passage de la boucle, on ajoute au second polynôme la dérivée du monôme du premier polynôme sur lequel on pointe. On passe ensuite le pointeur sur l'élément suivant. Une fois sortis de la boucle, on renvoie le second polynôme ainsi formé.

16. creerMonomeInteraction : Ø → Monome

Cette fonction ne prend aucun paramètre et renvoie un objet de type Monome. Voici le prototype de la fonction :

```
Monome creerMonomeInteraction ();
```

La fonction crée, en interaction avec l'utilisateur, un nouveau monôme. Elle est l'application pratique de la fonction permettant de créer un monôme. On demande simplement à l'utilisateur d'entrer une valeur pour le degré ainsi qu'une valeur pour le coefficient. On stocke ces valeurs et on crée un monôme associé à ces dernières. On renvoie ensuite le monôme ainsi créé.

17. creerPolynomeInteraction : Ø → Polynome

Cette fonction ne prend aucun paramètre et renvoie un objet de type Polynome. Voici le prototype de la fonction :

```
Polynome creerPolynomeInteraction ();
```

La fonction crée, en interaction avec l'utilisateur, un nouveau polynôme, en lui demandant d'entrer un à un les monômes qui le composeront. Pour ce faire, on initialise un nouveau polynôme, et on commence par une première acquisition du premier monôme, en supposant que l'utilisateur n'a pas d'utilité à créer interactivement un polynôme vide. Une fois le monôme créé par interaction, on demande à l'utilisateur s'il veut en ajouter d'autres ou pas. On ré-exécute ces instructions jusqu'à ce que l'utilisateur réponde non. A ce moment là, on lui annonce que son polynôme à bien été créé, et on renvoie ce polynôme qui vient d'être généré.

IV. Description du programme principal

Le programme principal a principalement pour but de gérer la communication avec l'utilisateur, et de lui permettre d'exploiter les sous-programmes prévus à cet effet.

On met donc tout d'abord en place un menu qui permettra à l'utilisateur de prendre en compte des différentes fonctionnalités mises à sa disposition dans ce programme. Elles sont au nombre de 8, sans compter l'option quitter le programme : créer un nouveau polynôme, supprimer un polynôme, ajouter un monôme à un polynôme, supprimer un monôme à un polynôme, additionner deux polynômes, multiplier deux polynômes, dériver un polynôme et enfin afficher un polynôme.

Une fois que l'utilisateur a fait son choix, les différentes instructions s'exécutent, et il peut arriver dans certains cas que l'on demande à l'utilisateur s'il veut conserver ou pas le nouveau polynôme issu des manipulations. Cela offre plus de liberté à l'utilisateur qui pourrait en avoir encore besoin pour d'autres applications. On lui demande alors de choisir un emplacement de mémoire, ce dernier étant limité à 10 polynômes maximum. On aurait pu rappeler à chaque fois quels sont les polynômes actifs ou utilisés depuis le départ, mais cela risque de surcharger inutilement l'écran. L'utilisateur doit donc organiser lui-même les polynômes, au risque d'écraser le polynôme existant déjà à une place. On pourrait imaginer un système de sécurité rappelant avant chaque écrasement le polynôme présent à cet endroit et seulement ensuite on confirmerait la mise en mémoire.

Par ailleurs, on choisi dans ce projet de nettoyer l'écran après chaque manipulation pour que l'utilisateur puisse rapidement repérer où se trouve le menu principal et donc apporter un plus en terme de confortabilité et d'ergonomie dans l'utilisation du programme. Pour cela on utilise une commande différente en fonction du système d'exploitation. On glisse la commande à la fin de chaque boucle associée à un choix, afin de nettoyer l'écran après la fin des instructions en rapport avec ce choix. Ces commandes sont *system("cls") ;* pour Windows et *system("clear") ;* pour Unix.

Il apparut aussi important de permettre à l'utilisateur d'accéder instantanément à l'affichage du polynôme issu d'une addition ou d'une dérivation. Afin de laisser le temps à l'utilisateur de récupérer visuellement ce polynôme, on place une temporisation. En effet, tant que l'utilisateur n'appuie pas sur une touche, l'écran se fige. Cette astuce est aussi présente dans la fonctionnalité qui affiche les polynômes : elle est d'ailleurs indispensable pour laisser le temps à l'utilisateur de l'apercevoir, la vitesse d'exécution étant très rapide.

V. Conclusion – Bilan

Ce projet nous a permi de mettre en pratique différentes connaissances accumulées au cours de cette unité de valeur, que ce soit en termes de programmation en langage C ou en compilation sous système Unix.

Nous avons appris à implémenter des formules et opérations mathématiques relatives aux polynômes en langage C. Par ailleurs, nous avons dû concevoir notre propre Makefile, facilitant la compilation sous Unix. L'utilisation du débuggeur du terminal s'est parfois avérée très utile pour dénicher la source de certains disfonctionnements, et son utilisation n'aurait pût être possible sans l'initiation apportée lors des séances de TP.

Enfin, en rédigeant ce rapport et en commentant rigoureusement le code au fur et à mesure de son écriture, nous avons assimilé des compétences et des réflexes qui nous seront très utiles pour nos futures études et travaux. On ne peut sortir que plus cultivé et rigoureux de cette expérience. On peut dire, au final, que ce premier projet sérieux nous a mi dans les bons rails, pour la suite de nos études.