RAPPORT DU MINI-PROJET C++

Jérôme Nicolas - Alexandre Ravaux

1 - Représentation creuse d'un polynôme générique et opérations

Nous avons utilisé une map pour représenter les coefficients comme conseillé.

```
1  map<int, T>map;
```

La clé de la map (qui est un int) représente **le degré** et la valeur qui lui est associée le **coefficient**. Par exemple `map[0]=1;` signifie que pour x^0 le coefficient est 1.

Vous pouvez créer un polynôme grâce au constructeur par défaut.

```
1  Polynome<T>monPolynome;
```

Vous pouvez également le créer en passant en paramètre un tableau de type T ou encore une map<int,T>.

```
1  Polynome<T>monPolynome(myTab, tailleTab);
2  Polynome<T>monPolynome(myMap);
```

Enfin, il existe un constructeur de copie.

```
1  Polynome<T>copiePolynome(monPolynome);
```

Grâce à la surcharge du flux stream, il est possible d'afficher le polynôme avec cout.

```
1  std::cout<<monPolynome<<std::endl;
```

Nous avons fait également la surcharge des opérateurs courants : l'addition, la soustraction et la multiplication. La multiplication avec un scalaire marche également.

Nous avons également écrit la division euclidienne, mais il se peut qu'elle ne soit pas tout à fait bonne (nous n'avons pas fait beaucoup de tests en interne pour la division).

N.B : T est un template, vous pouvez mettre une type tel que int ou double, bien entendu. Nous avons utilisé le type int dans le main.cpp (sauf pour la saisie), c'est un choix arbitraire.

2 - L'interpréteur

L'interpréteur est **fonctionnel**.

```
1 Polynome<double>monPolynomeSaisi(myTab, tailleTab);
2 monPolynomeSaisi.saisirPolynome();
```

Pour que la saisie soit effective, il faut écrire le polynôme de cette façon : a0x^0+a1x^1+…+anx^n
La fonction récupère en entrée ce que vous venez d'écrire avec `getline(cin, str)` et stocke cela dans une chaine de caractère `str`. Avant de procéder à l'analyse et parsing de la chaine de caractère, on enlève tout espace dans la chaîne pour simplifier l'analyse et le parsing. Enfin, on passe à l'analyse et parsing, ce qui consiste à découper la chaîne de caractère et stocker les différents coefficients. **Du coup, si vous écrivez x^2 par exemple cela ne marchera pas, il faut bien écrire le coefficient ainsi : 1x^2.**

3- Gestion des erreurs (Exceptions)

Nous avons ajouter un try{}catch(){} pour l'interpréteur.

Conclusion

Le projet a permis de consolider les bases en C++ mais également d'appréhender des notions plus complexes telle que la notion de surcharge des opérateurs. Cependant, nous avons rencontré des difficultés sur la gestion d'erreurs en C++.

Un tableau bilan est disponible en fin de rapport.

Tableau bilan

Tâche	État
Constructeur par défaut	✓
Constructeur avec un tableau de coefficients en argument	✓
Constructeur avec une map en argument	✓
Constructeur de copie	✓
Représentation du polynôme	✓
Surcharge du +	✓
Surcharge du -	✓
Surcharge du *	✓
Surcharge du /	≈
Surcharge du +=	✓
Surcharge du -=	✓
Surcharge du *=	✓
Surcharge du /=	≈
Surcharge du []	✓
Evaluer le polynôme	✓
Dérivée du polynôme	✓
Composition du polynôme	≈
Interpréteur	✓
Exceptions	✗

✓ : fait ≈ : fait mais pas tout à fonctionnel ✗ : pas fait