	Algorithmique et programmation séquentielle	
	ITI1	<i>Paquetages et utilisation (code de Reed-Solomon)</i>
	hepia, HES-SO//Genève	Laboratoire Ada

Préambule

Ce travail pratique est organisé en trois parties traitant de la notion de paquetage. Les parties sont les suivantes :

1. Implémentation d'un paquetage de fractions et d'un programme de test.
2. Implémentation d'un paquetage de polynômes à coefficients fractionnaires et d'un programme de test.
3. Utilisation des paquetages de fractions et de polynômes pour une implémentation d'un code de Reed-Solomon.

Recommandation

Vous êtes fortement encouragé à aller au libre service pour obtenir des conseils et de l'aide pour avancer votre TP.

Rendu

Le listing du code doit être rendu sur papier. **Veuillez imprimer deux pages par feuille.** Le code doit être bien **indenté, commenté et modularisé** avec des fonctions et procédures.

Les codes sources doivent être rassemblés dans une **archive .zip à votre nom** qui sera déposée sur le site du cours dans <https://cyberlearn.hes-so.ch>

Plus précisément, l'étudiant **Laurent Sampaio** mettra **uniquement** 7 fichiers nommés :


```
gestion_fractions.ads
gestion_fractions.adb
calcul_fractions.adb
gestion_polynomes.ads
gestion_polynomes.adb
calcul_polynomes.adb
reed_solomon.adb
```

dans un **répertoire** nommé **laurent_sampaio** qui sera lui même zippé en un **fichier** nommé **laurent_sampaio.zip**. C'est ce fichier zippé qui devra être déposé dans <https://cyberlearn.hes-so.ch>

Attention ni espaces, ni accents, ni majuscules dans les noms !

Sous peine de sanction, vous devez respecter toutes ces spécifications.

Ce travail pratique est noté. L'évaluation sera faite sur la base du listing et d'une interrogation orale lors de laquelle vous expliquerez le travail réalisé.

	Algorithmique et programmation séquentielle	
	ITI1	<i>Paquetage de gestion de fractions</i>
	hepia, HES-SO//Genève	Laboratoire Ada

Buts

- Mettre en œuvre la notion d'article
- Utiliser des procédures et fonctions (récursives)
- Créer un paquetage
- Définir des opérateurs
- Déclarer, lever et traiter une exception

Enoncé

Il s'agit d'écrire un paquetage `Gestion_Fractions` pour la gestion de fractions. Ce paquetage offrira notamment comme fonctionnalités la saisie, l'affichage, l'addition, la soustraction, la multiplication et la division des fractions. Les fractions devront toujours être stockées sous forme irréductible. Dans le paquetage, déclarer aussi une exception `DIV_PAR_ZERO`.

Ecrire ensuite un programme `calcul_fractions.adb` qui utilise le paquetage `Gestion_Fractions`, et qui affiche le résultat d'un calcul passé en argument sur la ligne de commande (comme l'addition de deux fractions). Ce programme doit traiter l'exception `DIV_PAR_ZERO` quand une division par zéro est propagée.

Cahier des charges

Le paquetage `Gestion_Fractions` sera constitué de

- un article `T_Fraction` composé de 2 champs : le numérateur et le dénominateur ;
- 3 procédures :
 - une procédure `Get` qui lit une fraction au clavier en gérant les erreurs de saisie p. ex. via les exceptions et la récursivité ;
 - une procédure `Put` qui affiche une fraction à l'écran ;
 - une procédure `Reduire` qui rend une fraction irréductible ;
- 10 fonctions :
 - un opérateur `+` qui effectue l'addition de 2 fractions et qui retourne une fraction irréductible ;
 - un opérateur `-` qui effectue la soustraction de 2 fractions et qui retourne une fraction irréductible ;
 - trois opérateurs `*` qui effectuent la multiplication soit de 2 fractions, soit 1 entier et 1 fraction, soit 1 fraction et 1 entier, et qui retournent une fraction irréductible ;
 - trois opérateurs `/` qui effectuent la division soit de 2 fractions, soit 1 entier et 1 fraction, soit 1 fraction et 1 entier, et qui retournent une fraction irréductible ;
 - un opérateur `**` qui met une fraction à une puissance entière, et qui retourne une fraction irréductible ;
 - une fonction `PGCD` qui calcule le P.G.C.D. de 2 nombres entiers positifs, celle-ci sera appelée par la procédure `Reduire` ;
- une exception `DIV_PAR_ZERO` qui sera levée en cas de division par zéro dans la fonction `/`.

Le type `T_Fraction`, l'exception `DIV_PAR_ZERO` ainsi que les entêtes des procédures et des fonctions constituent le fichier de spécification `gestion_fractions.ads` du paquetage. L'intégralité des procédures et fonctions se trouvent dans le fichier de corps du paquetage `gestion_fractions.adb`.

Le programme test `calcul_fractions.adb` utilisera, comme dit dans l'énoncé, le paquetage `Gestion_Fractions` en mettant dans la clause de contexte :

```
with Gestion_Fractions; use Gestion_Fractions;
```

Le lancement du programme avec deux fractions (ou une fraction et un entier selon l'opération) et une opération (+, -, ×, /, p) en argument sur la ligne de commande doit afficher le résultat du calcul. Sinon si on passe en argument PGCD et deux entiers positifs, alors le PGCD de ces deux nombres s'affiche.

Rien d'autre ne devra être affiché.

Par exemple :

```
~> ./calcul_fractions 3 10 + 8 15
      5 6
~> ./calcul_fractions 3 10 x 8 15
      4 25
~> ./calcul_fractions 3 x 8 11
      24 11
~> ./calcul_fractions 3 10 x 8
      12 5
~> ./calcul_fractions 3 10 p -2
      100 9
~> ./calcul_fractions 15 10 PGCD
      5
```

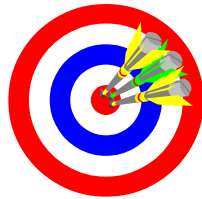
Remarques

- Afficher et saisir une fraction sous forme d'une paire d'entiers
- Les noms des fichiers du paquetage doivent être en **MINUSCULES**
- La compilation de `calcul_fractions.adb` entraîne automatiquement celle du paquetage
- Pour compiler seulement le paquetage : `gcc -c gestion_fractions.adb`
- Pour déclarer une exception : `DIV_PAR_ZERO : Exception;`
- Pour lever une exception : `raise DIV_PAR_ZERO;`
- Pour traiter une exception, on introduit avant l'instruction `end;` de la fin d'un bloc (par exemple la fin d'une fonction, d'une procédure, d'une boucle ou d'un `declare`), un `traitement exception` :

```
exception
when EXCEPT_1 => traitement_1;
when EXCEPT_2 => traitement_2;
...
when others => autre_traitement;
```

- Pour la fonction PGCD, utiliser l'algorithme de division d'Euclide (ou bien sa version récursive) :

<pre>Pour N et M deux nombres entiers positifs: Tant que le reste de N / M est différent de zéro faire N prend la valeur de M M prend la valeur du reste de N / M Fin de tant que Le PGCD de N et M vaut M</pre>
--

	Algorithmique et programmation séquentielle	
	ITI1	<i>Paquetage de gestion de polynômes</i>
	hepia, HES-SO//Genève	Laboratoire Ada

Buts

- Mettre en œuvre la notion de tableau non-contraint et d'article à discriminant
- Utiliser des procédures et fonctions
- Créer un paquetage
- Définir des opérateurs
- Evaluer un polynôme en une valeur

Enoncé

Il s'agit d'écrire un paquetage `Gestion_Polynomes` pour la gestion de polynômes à coefficients fractionnaires. Ce paquetage offrira comme fonctionnalités la saisie, l'affichage, l'addition, la soustraction, la multiplication et la division des polynômes (quotient et reste), ainsi que l'évaluation sur une fraction.

Ecrire ensuite un programme `calcul_polynomes.adb` qui utilise le paquetage `Gestion_Polynomes`, et qui affiche le résultat d'un calcul passé en argument sur la ligne de commande (comme l'addition de deux polynômes).

Cahier des charges

Le paquetage `Gestion_Polynomes` sera constitué de

- un article à discriminant `T_Polynome` défini de la manière suivante

```

subtype T_Degre is Natural range 0..10000;
type T_Coeff is array(T_Degre range <>) of T_Fraction;
type T_Polynome(Degre : T_Degre := 0) is record
    Coeff : T_Coeff(0..Degre);
end record;

```
- 2 procédures
 - une procédure `Get` qui lit un polynôme au clavier ;
 - une procédure `Put` qui affiche un polynôme à l'écran ;
- 8 fonctions
 - un opérateur `+` qui effectue l'addition de 2 polynômes, et qui retourne un polynôme ;
 - un opérateur `-` qui effectue la soustraction de 2 polynômes, et qui retourne un polynôme ;
 - trois opérateurs `*` qui effectuent la multiplication soit de 2 polynômes, soit 1 polynôme et 1 fraction, soit 1 fraction et 1 polynôme, et qui retournent un polynôme ;
 - un opérateur `/` qui effectue la division de 2 polynômes, et qui retourne un polynôme (le quotient de la division) ;
 - une fonction `Reste` qui calcule le reste de la division de 2 polynômes, et qui retourne un polynôme (le reste de la division) ;
 - une fonction `Eval` qui évalue un polynôme sur une fraction (de préférence via la méthode de Hörner), et qui retourne une fraction ;
 - une fonction `Alloc_Polyn` avec en paramètre `n` de type `T_Degre`, et qui retourne le polynôme x^n .

Le type `T_Polynome` et les types qui permettent de le définir, ainsi que les entêtes des procédures et des fonctions constituent le fichier de spécification `gestion_polynomes.ads` du paquetage. L'intégralité des procédures et fonctions se trouvent dans le fichier de corps `gestion_polynomes.adb` du paquetage.

Le programme de test `calcul_polynomes.adb` utilisera le paquetage `Gestion_Polynomes` en mettant dans la clause de contexte :

```
with Gestion_Polynomes; use Gestion_Polynomes;
```

Un polynôme doit toujours avoir son dernier coefficient non-nul (celui correspondant au degré du polynôme), sauf si celui-ci est une constante (c.-à-d. de degré 0). Les différentes procédures et fonctions du paquetage doivent le garantir lorsque c'est nécessaire.

Le programme lancé avec en argument sur la ligne de commande deux polynômes et une opération (+, -, ×, /, r, e) doit afficher le résultat du calcul.

Rien d'autre ne devra être affiché.

Par exemple, pour l'addition de $\frac{1}{2}x^2 + \frac{13}{4}x^3 + \frac{7}{3}x^4$ et $-\frac{3}{2}x + \frac{15}{7}x^2$

```
~> ./calcul_polynomes 1 2 0 1 -2 1 13 4 7 3 + -3 2 7 2 15 7
      -1 1 7 2 1 7 13 4 7 3
```

et, pour l'évaluation du $-\frac{3}{2}x + \frac{15}{7}x^2$ au point $x=1$

```
~> ./calcul_polynomes -3 2 7 2 15 7 e 1 1
      29 7
```

et pour le reste de la division de $\frac{1}{2}x^5 + \frac{5}{2}x^2 - 2x^2 + \frac{13}{4}x^3 + x^4$ par $-\frac{3}{2}x + x^2$

```
~> ./calcul_polynomes 1 2 5 2 -2 1 13 4 1 1 r -3 2 7 2 1 1
      17 16 13 16
```

Attention : le polynôme est affiché en commençant par le coefficient de degré 0 jusqu'à celui du degré du polynôme.

Remarque : valeur d'un polynôme en un point

Soit $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$ un polynôme et x_0 un nombre. Le calcul de $P(x_0)$ laisse à penser qu'il faut calculer chacune des puissances de x_0 , multiplier celle-ci par son coefficient a_k , puis faire la somme du tout.

Si on calcule x_0 en multipliant successivement x_0 par lui-même, le nombre de produits nécessaire est alors de $n+(n-1)+\dots+2+1 = n(n+1)/2$, quantité qui croît comme n^2 , le carré du degré du polynôme.

On pourrait améliorer la vitesse du calcul de x_0 par une méthode d'exponentiation rapide (dénommée multiplication indienne), qui réduit le temps de calcul de $P(x_0)$ à $n \cdot \log_2(n)$.

La **méthode de Horner** consiste à améliorer encore plus ce résultat en faisant le calcul :

$$P(x_0) = (((...((a_n x_0 + a_{n-1}) x_0 + a_{n-2}) x_0 + \dots) x_0 + a_1) x_0 + a_0$$

Le nombre de produits est alors réduit à n , de sorte que le temps de calcul d'une fonction polynomiale en un point a est seulement proportionnel au degré du polynôme. La méthode consiste donc à multiplier le 1er coefficient par x_0 et à lui ajouter le 2ème coefficient. On multiplie alors le nombre obtenu par x_0 et on lui ajoute le 3ème coefficient, etc. Elle peut s'organiser avec un tableau dans lequel

chaque case de la 2ème ligne est obtenue en multipliant le coefficient de la case de gauche par x_0 et en lui ajoutant le coefficient de la case du dessus.

Coefficients de P	a_n	a_{n-1}	a_{n-2}	...	a_1	a_0
Facteur de x_0	a_n	$a_n x_0 + a_{n-1}$	$(a_n x_0 + a_{n-1}) x_0 + a_{n-2}$...	q_0	$P(x_0) = q_0 x_0 + a_0$

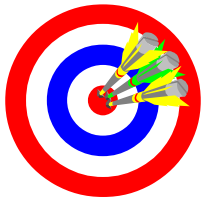
Exemple pratique : calcul de $4x^3 - 7x^2 + 3x - 5$ pour $x = 2$.

Coefficients de P	4	-7	3	-5
Facteur de x_0	4	$8 - 7 = 1$	$2 + 3 = 5$	$P(x_0) = 10 - 5 = 5$

Cette méthode permet aussi de faire une conversion rapide d'un nombre écrit en base x_0 en écriture en base 10. En effet, si un nombre s'écrit $a_n a_{n-1} \dots a_0$ en base x_0 , ce nombre vaut $a_n x_0^n + a_{n-1} x_0^{n-1} + \dots + a_0$ en base 10.

Exemple pratique : écriture en base 10 du nombre hexadécimal 0xDA78 (en base 10 on a D = 13 et A = 10).

Coefficients de P	13	10	7	8
Facteur de 16	13	$13 \times 16 + 10 = 218$	$218 \times 16 + 7 = 3495$	$0xDA78$ $= 3495 \times 16 + 8 = 5592$

	Algorithmique et programmation séquentielle	
	ITI1	<i>Code de Reed-Solomon</i>
	hepia, HES-SO//Genève	Laboratoire Ada

Buts

- Implémentation d'un code de Reed-Solomon
- Interpolation polynomiale de Lagrange
- Utilisation des paquetages créés précédemment

Enoncé

On peut résumer un code de Reed-Solomon de la manière suivante. On considère une liste de k octets (c.-à-d. des nombres entre 0 et 255) à transmettre dans un canal bruité. L'encodage consiste à construire le polynôme d'interpolation P passant par les points formés de la paire indice-valeur de cette liste, puis à ajouter un nombre fixé n de valeurs $P(k), P(k+1), \dots, P(k+n-1)$ à la liste pour la rendre robuste. La liste augmentée de $k+n$ octets est ensuite transmise à travers un canal bruité qui introduit des erreurs, c.-à-d. certains octets sont modifiés. A la réception de la liste bruitée de $k+n$ octets, on enlève le bruit en considérant chaque sous-ensembles de k éléments de cette liste avec son polynôme d'interpolation associé. On comptabilise le nombre d'apparitions d'un polynôme et on retient celui qui est majoritaire (s'il y a $\leq \lfloor n/2 \rfloor$ erreurs, alors on retrouve le polynôme d'interpolation initial P). On peut ainsi récupérer la liste des $k+n$ octets avant perturbation, et ensuite la liste initiale de k octets.

Il s'agit d'écrire un programme `reed_solomon.adb` qui implémente un code de Reed-Solomon. Pour cela, vous devez réutiliser le paquetage de gestion de polynômes à coefficients fractionnaires, créé précédemment. Dans la procédure principale, on fixe les valeurs de n et k . Ensuite, on entre k nombres entre 0 et 255 (la liste d'octets à envoyer) et on affiche le polynôme d'interpolation associé. Puis on effectue successivement les étapes d'encodage (c.-à-d. l'ajout de n points supplémentaires), de bruitage, de débruitage et finalement de décodage telles que décrites précédemment. A chaque étape, on affiche les listes d'octets résultantes. Pour le débruitage, il faut aussi afficher le tableau des polynômes d'interpolation associés aux sous-ensembles de k éléments, ainsi que le nombre d'occurrences de ces polynômes.

Cahier des charges

En plus de la procédure principale, vous devez au moins écrire :

- une fonction (récursive) produit qui retourne un polynôme égal au produit $(x-a_1)(x-a_2)\dots(x-a_n)$;
- une fonction `interpolate` qui calcule le polynôme d'interpolation de Lagrange d'un ensemble de points.

Le programme aura deux modes de fonctionnement. S'il est lancé sans argument sur la ligne de commande, il s'exécute en interactif. Par contre, s'il est lancé avec une liste d'entiers $x_1, y_1, x_2, y_2, \dots, x_n, y_n$ en argument sur la ligne de commande, comme par exemple : `./reed_solomon 1 4 -2 7 8 9 -1 2`

alors le programme **affichera uniquement** le polynôme d'interpolation par les points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ dans le format du package de gestion des polynômes.

Références

https://fr.wikipedia.org/wiki/Code_de_Reed-Solomon
http://fr.wikipedia.org/wiki/Polynôme_de_Lagrange