Membres du binôme :	*Groupe :*

Séance I

Nombres premiers

1 Introduction

Il s'agit durant les deux premières séances de concevoir une classe statique permettant les calculs de base sur les nombres premiers :

- Génération des nombres premiers inférieurs à une valeur donnée
- Vérification si deux nombres donnés sont premiers entre eux
- Décomposition en produit de facteurs premiers
- Calcul de l'indicateur d'Euler

2 Création du projet et de la classe *Premier*

1. Ouvrir Microsoft Visual Studio et créer un nouveau projet .Net Core en mode console.
2. Ajouter une nouvelle classe statique au projet. Lui donner le nom de **Premier**.

▷ **Question 1** ☐ Pourquoi créer une classe *statique* ? Quel est le mot clé en C# qui permet de créer de telles classes ?

>

3 Le crible d'Eratosthène

La méthode du **crible d'Eratosthène** fournit les nombres premiers inférieurs à un entier n donné.

Cette méthode a été imaginée au IIIe siècle av. J.-C. par le mathématicien grec Eratosthène, qui a été par ailleurs directeur de la bibliothèque d'Alexandrie.

On commence par écrire la liste des entiers jusque n.

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad ... \quad n-1 \quad n$$

On biffe le 1, qui n'est pas un nombre premier. Le premier nombre non biffé, 2, est un nombre premier.

$$\not{1} \quad 2 \quad 3 \quad 4 \quad 5 \quad ... \quad n-1 \quad n$$

Ensuite, on biffe tous les multiples de 2 : ils ne sont pas premiers car ils sont divisibles par 2. Le premier nombre non biffé, 3, est un nombre premier.

$$\not{1} \quad 2 \quad 3 \quad \not{4} \quad 5 \quad \not{6} \quad 7 \quad \not{8} \quad ... \quad n-1 \quad n$$

On recommence le procédé en biffant tous les multiples de 3.

$$\not{1} \quad 2 \quad 3 \quad \not{4} \quad 5 \quad \not{6} \quad 7 \quad \not{8} \quad \not{9} \quad ... \quad n-1 \quad n$$

On réitère le procédé jusqu'à la partie entière de $\sqrt{n}$. En effet, si un entier non premier est strictement supérieur à $\sqrt{n}$, alors il a au moins un diviseur inférieur à $\sqrt{n}$ et a donc déjà été biffé.

Par exemple, pour $n = 10$, on itère jusque l'entier 3 (car $\sqrt{10} \simeq 3.16$) et on trouve les nombres premiers 1, 3, 5 et 7 :

$$\not{1} \quad 2 \quad 3 \quad \not{4} \quad 5 \quad \not{6} \quad 7 \quad \not{8} \quad \not{9} \quad \not{10}$$

▷ **Question 2** ☐ Quel est le type de donnée le plus adapté pour n ? Est-ce suffisant ? Pourquoi ? Dans la suite de cette étude, on se contentera de travailler sur des entiers (type *int*).

```
>
```

▷ **Question 3** ☐ Programmer cette méthode au sein de la classe *Premier*. Téléverser ce code sur l'ENT. Indication : enquêter du coté de l'instruction C# *Math.Sqrt*.

▷ **Question 4** ☐ Cette méthode est-elle adaptée au chiffre RSA ? Pourquoi ?

```
>
```

Jeux de test. Les nombres premiers inférieurs à 100 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

▷ **Question 5** ☐ A l'aide de la méthode ci-dessus, citer les nombres premiers compris entre ... et ...

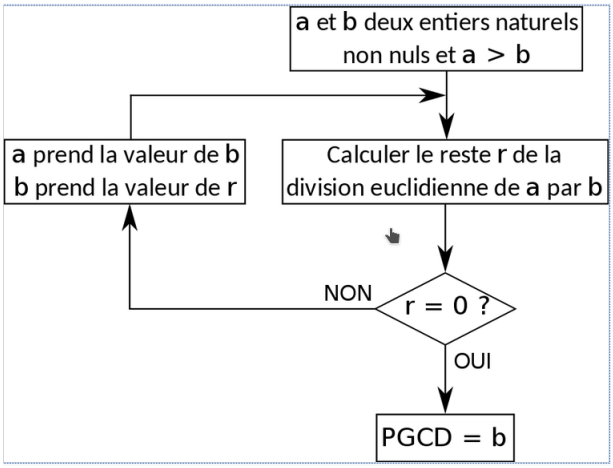
```
>
```

4 Nombres premiers entre eux

Deux nombres sont premiers entre eux s'ils n'ont que 1 comme diviseur commun. Autrement dit, leur plus grand diviseur commun est 1. Déterminer si deux nombres sont premiers entre eux revient

donc à calculer leur PGCD. Dans cette étude, on utilise l'algorithme d'Euclide (mathématicien grec qui vécut vers -300 av. J.-C.).

Le fonctionnement de cet algorithme est illustré par la figure suivante.



▷ **Question 6** □ Au sein de la classe *Premier*, programmer une méthode statique qui implémente l'algorithme d'Euclide pour déterminer si deux entiers donnés sont premiers entre eux. *Si ce travail n'est pas terminé en séance, le réaliser en autonomie !*

Jeux de test. 79 et 3337 sont premiers entre eux. 11 et 44 ne sont pas premiers entre eux.

Séance II

Calcul de l'indicateur d'Euler

1 Décomposition en produit de facteurs premiers

La décomposition en produit de facteurs premiers consiste à écrire un entier naturel non nul n sous forme d'un produit de nombres premiers.

Exemple. $45 = 3^2 \times 5$

Algorithme. Il s'agit de balayer par ordre croissant la liste des nombres premiers en testant si le nombre premier p divise n. Si oui, on recommence l'algorithme pour n/p, en ne testant que les diviseurs premiers encore envisageables. On s'arrête quand on aboutit sur 1.

▷ **Question 7** ☐ Ajouter à la classe *Premier* une méthode statique réalisant la décomposition en produit de facteurs premiers du nombre passé en paramètre. Téléverser ce code sur l'ENT.

Indications. L'utilisation de listes (classe générique *List<>*) peut être judicieuse. On donne ci-dessus un exemple d'utilisation d'une liste de réel au format *double*.

```
List<double> maListe = new List<double>(); // Création d'une liste vide maListe.
maListe.Add(3.14);                         // Ajout d'un élément à la liste :
                                           // maListe vaut {3.14}
maListe.Add(-6.4);                         // Ajout: maListe vaut {3.14 ; -6.4}

foreach(double val in maListe)             // Affichage du contenu de la liste
   Console.WriteLine(val);                 // S'affiche à l'écran: 3.14  -6.4
```

▷ **Question 8** ☐ A l'aide de la méthode ainsi rédigée, donner la décomposition en produit de facteurs premiers de ...

```
>
```

2 Indicateur d'Euler

On rappelle que, si $n = p^u \times q^v \times ... \times r^w$, avec p, q, ..., r premiers, alors

$$\varphi(n) = p^{u-1} \times (p-1) \times q^{v-1} \times (q-1) \times ... \times r^{w-1} \times (r-1)$$

▷ **Question 9** ☐ Ajouter à la classe *Premier* une méthode statique qui retourne l'indicateur d'Euler de l'entier passé en paramètre.

Jeux de test. $\varphi(10403) = 10200$, $\varphi(3220) = 1056$

▷ **Question 10** ☐ A l'aide de la méthode ci-dessus, indiquer l'indicateur d'Euler de ...

```
>
```

Séance III

Puissance modulo n et chiffrement/déchiffrement d'un entier

1 Élévation à la puissance modulo n

1.1 Rappels préliminaires

On rappelle que :

$$x^{a+b} = x^a \times x^b$$

$$(a \times b) \mod n = (a \mod n \times b \mod n) \mod n$$

1.2 Exponentiation modulaire

On a vu en cours que l'élévation à la puissance pouvait très vite entraîner un dépassement de la capacité de calcul des langages informatiques classiques. Par exemple, C# est incapable de calculer l'exemple du cours $79^{1055} \mod 3220$ pour déterminer la clé de déchiffrement d.

Cette méthode directe ne profite pas de la structure d'anneau commutatif du domaine dans lequel on travaille : les entiers modulo n. En combinant ceci avec le principe d'exponentiation rapide, on simplifie drastiquement les calculs.

Exemple. Considérons le calcul $4^{13} \mod 497$.

Dans un premier temps, on convertit 13 en binaire :

$$13 = (1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Donc :

$$\begin{aligned}
4^{13} \mod 497 &= 4^{1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0} \mod 497 && (1)\\
&= 4^{1\times 2^3} \times 4^{1\times 2^2} \times 4^{0\times 2^1} \times 4^{1\times 2^0} \mod 497 && (2)\\
&= 4^{1\times 8} \times 4^{1\times 4} \times 4^{0\times 2} \times 4^{1\times 1} \mod 497 && (3)\\
&= (4^{1\times 8} \mod 497 \times 4^{1\times 4} \mod 497 \times 4^{0\times 2} \mod 497 \times 4^{1\times 1} \mod 497) \mod 497 && (4)\\
&= (4^8 \mod 497 \times 4^4 \mod 497 \times 4^0 \mod 497 \times 4^1 \mod 497) \mod 497 && (5)\\
&= (65536 \mod 497 \times 256 \mod 497 \times 1 \mod 497 \times 4 \mod 497) \mod 497 && (6)\\
&= (429 \times 256 \times 1 \times 4) \mod 497 && (7)\\
&= 439296 \mod 497 && (8)\\
&= 445 && (9)
\end{aligned}$$

Si on avait calculé directement, 4^{13} aurait donné 67108864. Bien que 4 n'ait qu'un chiffre et 13 deux, la valeur 4^{13} atteint une longueur de 8 chiffres ! Si on prend un nombre b à 77 chiffres décimaux, ce qui est un petit chiffre en cryptographie, b^{13} comporte 1304 chiffres, ce qui est énorme.

Attention. Cette technique donne des résultats incorrects si l'un des produits déborde avant la réduction modulo n. Telle quelle, elle ne peut donc être utilisée que pour des valeurs de n inférieures à 2^{16} en langage C.

Au niveau programmation, la ligne 4 décrit l'essence de l'algorithme. Si on remarque que $a^{2^{i+1}} = a^{2^i \times 2} = (a^{2^i})^2$, l'algorithme vient aisément.

▷ **Question 11** ☐ Programmer une méthode réalisant l'exponentiation modulaire. Indication : la méthode *Convert.ToString* permet de convertir un entier en notation binaire.

Jeux de test. $5^{11} \mod 14 = 3$, $17^{154} \mod 100 = 29$, $4^{13} \mod 497 = 445$

2 La classe RSA

2.1 Vérification de la validité de la clé de chiffrement et calcul de la clé de déchiffrement

▷ **Question 12** ☐ Ajouter au projet une classe **RSA** permettant de modéliser un système cryptographique RSA. Quels sont les attributs de cette classe ?

>

▷ **Question 13** ☐ Rappeler les deux conditions pour qu'une clé de chiffrement (e, n) soit valide.

>

▷ **Question 14** ☐ Concevoir un constructeur de cette classe qui, après avoir vérifié si les éléments fournis par l'utilisateur sont valides, calcule la clé de déchiffrement d. Rappeler dans le cadre ci-dessous la formule mathématique permettant d'obtenir d.

>

Jeux de test. Les triplets (p, q, e) suivants sont valides : (47, 71, 79), (5, 17, 5) ; d vaut respectivement 1019 et 13. Ceux-ci sont non valides : (4, 11, 3), (3, 11, 40).

▷ **Question 15** ☐ La clé (... , ... , ...) est-elle valide ? Si oui, quelle est la clé de déchiffrement ?

>

2.2 Chiffrement et déchiffrement d'un entier

2.2.1 Chiffrement d'un entier

▷ **Question 16** ☐ Concevoir une méthode qui chiffre l'entier passé en paramètre.

Jeux de test. Avec le triplet (47, 71, 79), l'entier 61 est chiffré en 1701 et l'entier 215 en 1435.

▷ **Question 17** ☐ A quelle valeur est chiffré l'entier ... si on utilise la clé (5, 17, 5) ?

```
>
```

2.2.2 Déchiffrement d'un entier

▷ **Question 18** ☐ Concevoir une méthode qui déchiffre l'entier passé en paramètre.

Jeux de test. Avec le triplet (47, 71, 79), l'entier 1701 est déchiffré en 61 et l'entier 1435 en 215.

▷ **Question 19** ☐ A quelle valeur est déchiffré l'entier ... si on utilise la clé (5, 17, 5) ?

```
>
```

▷ **Question 20** ☐ Téléverser sur l'ENT le code de la classe **RSA**.

Séance IV

Méthodes liées au chiffrement d'une chaîne

1 Traduction chaîne de caractères vers suite de nombres

Pour chiffrer/déchiffrer des messages sous forme de chaîne de caractères, on se propose de représenter les 26 lettres majuscules de l'alphabet latin par des nombres à deux chiffres compris entre 01 et 26, correspondant à leur rang dans l'alphabet ($A \to 01, B \to 02..., Z \to 26$), l'espace étant représenté par 00.

Par exemple, la chaîne de caractères **FLORENCE** sera traduite comme suit :

```
F  L  O  R  E  N  C  E
06 12 15 18 05 14 03 05
```

▷ **Question 21** ☐ Donner un exemple de codage possible autre que le rang dans l'alphabet.

```
>
```

▷ **Question 22** ☐ Concevoir une méthode qui code une chaîne de caractères selon la méthode ci-dessus. Cette méthode est-elle contenue dans la classe RSA ? Argumenter. Téléverser ce code sur l'ENT.

```
>
```

Jeux de test. La chaîne de caractères **VIVE UNIX** est codée en 220922050021140924.

Indications.

- En C#, un caractère est représenté par son code ASCII.
  Ainsi, l'instruction `int i = 'A' + 2;` est valide : i contient $65 + 2 = 67$.
- S'il est appliqué sur deux chaînes de caractères, l'opérateur + concatène ces deux chaînes.
- La méthode *ToString* obtient la représentation chaîne de caractères de l'objet appelant. Ainsi, si un entier i vaut 67, l'instruction `string str = i.ToString();` affecte à *str* la chaîne de caractères « 67 ».

2 Découpage de la suite de chiffres en blocs

Ensuite, la chaîne traduite sous forme d'une suite de chiffres est découpée en blocs de longueur inférieure au nombre de chiffres de n. Par exemple, si n vaut 3337, les blocs seront de longueur 3 ($3 < 4$).

Ainsi, la chaîne de caractères **FLORENCE** sera ensuite traitée comme suit. Noter l'ajout de deux zéros au dernier bloc pour que les blocs aient tous la même taille.

```
F  L  O  R  E  N  C  E
06 12 15 18 05 14 03 05
061 215 180 514 030 500
61 215 180 514 30 500
```

▷ **Question 23** ☐ Pourquoi coder des blocs, et non pas directement chaque caractère du message clair un par un ?

```
>
```

▷ **Question 24** ☐ Concevoir une méthode réalisant ce découpage. La méthode pourra retourner une liste d'entiers (classe générique *List*<>). Téléverser ce code sur l'ENT.

Indication. La méthode *int.Parse* permet de convertir une chaine de caractères en l'entier équivalent. Par exemple, `int.Parse(''061'')` retourne l'entier 61.

Séance V

Méthodes liées au déchiffrement d'une chaîne

1 Traduction liste d'entiers vers chaîne de caractères

▷ **Question 25** □ Concevoir une méthode qui prend en entrée une liste d'entiers et un entier l, et qui retourne cette liste sous forme d'une chaîne de caractères de manière à ce que chaque entier soit traduit par une sous chaîne d'une longueur l. Téléverser ce code sur l'ENT.

Pour le cas de **FLORENCE**, on se situe à l'étape « de ceci vers cela ». Chaque entier est représenté par une sous-chaîne de longueur 3.

```
F  L  O  R  E  N  C  E
06 12 15 18 05 14 03 05
061  215  180  514  030  500    chiffrement
1701 1435 2645 1111 2604 442
-- transfert des blocs chiffrés --
1701 1435 2645 1111 2604 442    déchiffrement
  61  215  180  514   30 500    |de ceci
 061|215|180|514|030|500        |vers cela
 06 12 15 18 05 14 03 05 00
 F  L  O  R  E  N  C  E
```

Indication. On pourra enquêter sur le format **D** de la méthode *ToString*.

2 Traduction chaîne de caractères codée vers chaîne de caractères claire

▷ **Question 26** □ Concevoir une méthode qui prend en entrée une chaîne de caractères où chaque bloc de deux chiffres représente une lettre de l'alphabet par rapport à son rang ($01 \rightarrow A, 02 \rightarrow B, \ldots, 00 \rightarrow$ *espace*) et qui retourne la chaîne claire correspondante. Téléverser ce code sur l'ENT.

Pour le cas de **FLORENCE**, on se situe à l'étape « de ceci vers cela ».

```
-- transfert des blocs chiffrés --
1701 1435 2645 1111 2604 442    déchiffrement
  61  215  180  514   30 500
 061|215|180|514|030|500        |de ceci
 06 12 15 18 05 14 03 05 00
 F  L  O  R  E  N  C  E         |vers cela
```

Séance VI

Chiffrement et déchiffrement d'une chaîne de caractères

1 Chiffrement

▷ **Question 27** □ Réaliser une surcharge de la méthode de chiffrement développée dans la séance III qui réalise le chiffrement d'une chaîne de caractères à l'aide des méthodes développées durant la séance IV. La méthode pourra retourner une liste de blocs chiffrés (classe générique *List*<>). Téléverser cette méthode sur l'ENT.

Ainsi, la chaîne de caractères **FLORENCE** sera traitée comme suit.

```
F  L  O  R  E  N  C  E
06 12 15 18 05 14 03 05
061  215  180  514  030  500
1701 1435 2645 1111 2604 442   cryptogramme correspondant à FLORENCE
```

▷ **Question 28** □ Donner le cryptogramme correspondant au chiffrement avec la clé (47, 71, 79) de la chaîne suivante : ...

```
>
```

2 Déchiffrement

▷ **Question 29** □ Réaliser une surcharge de la méthode de déchiffrement développée durant la séance III permettant le déchiffrement d'une suite de blocs chiffrés. Utiliser les méthodes développées durant la séance V. Téléverser ce code sur l'ENT.

```
F  L  O  R  E  N  C  E
1701 1435 2645 1111 2604 442    chiffrement
-- transfert des blocs chiffrés --
1701 1435 2645 1111 2604 442    déchiffrement
 061  215  180  514  030 500
06 12 15 18 05 14 03 05 00
F  L  O  R  E  N  C  E
```

▷ **Question 30** □ A quel message clair correspond le cryptogramme suivant, chiffré avec la clé (47, 71, 79) ?

```
>
```