Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Consignes à lire impérativement !

L'examen est composé de **2 parties**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- Commencez par écrire vos **nom**, **prénom** et **section** (math, info, ...) sur chaque feuille, y compris les feuilles de brouillon.
- Laissez vos calculatrice, téléphone portable et notes de cours dans votre sac. Leur usage n'est **pas autorisé**. Pensez à éteindre votre téléphone portable !
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Utilisez pour vos réponses les **cadres** prévus à cet effet. Si davantage d'espace est nécessaire, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement où se situe le restant de la réponse.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).
- Toutes les feuilles (énoncé et brouillon) doivent être remises en fin d'examen.
- Vérifiez que vous avez répondu à toutes les questions (il y a **2 questions** dans cette partie).

Question 1 – Fraction irréductible (/3)

L'objectif de cette question est la conception d'un algorithme de simplification de fractions. Soit p et q deux nombres naturels non nuls. Simplifier la fraction $\frac{p}{q}$ consiste à déterminer p' et q' naturels premiers entre eux tels que $\frac{p}{q} = \frac{p'}{q'}$. On nomme $\frac{p'}{q'}$ fraction irréductible car il n'est pas possible de la simplifier, p' et q' n'ayant pas de diviseur commun.

Dans cet exercice, on considère que le numérateur et le dénominateur de la fraction à simplifier sont chacun fournis sous forme de leurs facteurs premiers. Par exemple, la fraction $\frac{45}{6}$ pourrait être fournie comme $\frac{3\times5\times3}{3\times2}$. La fraction irréductible correspondante est $\frac{15}{2}$.

Ce qui vous est demandé : Ecrire une méthode `reduce` qui retourne la simplification d'une fraction $\frac{p}{q}$, sous la forme d'une fraction irréductible. La méthode prend deux arguments `fp` et `fq` qui sont respectivement les listes de facteurs premiers de p et q. Ces facteurs premiers peuvent être fournis dans un ordre quelconque, mais ne peuvent être vides. De plus, on considère que tout élément de fp et fq est strictement positif. La méthode `reduce` doit retourner la fraction sous forme d'une paire d'entiers (numérateur et dénominateur). La signature de `reduce` est partiellement montrée ci-dessous. Il vous est demandé d'en fournir l'implémentation et d'en compléter la signature.

```
public /* Type à déterminer */ reduce(int[] fp, int[] fq);
```

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1 **(méthode `reduce`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Question 2 – Ensemble ordonné (/7)

Cette question concerne une structure de données conservant un ensemble ordonné d'éléments. L'implémentation de cette structure de données repose sur un tableau de N listes L_i ($0 \leq i < N$), comme illustré à la Figure 1. La taille N du tableau est fixe. Les tailles N_i des listes varient en fonction du nombre d'éléments (N_{elts}) présents. Les éléments sont répartis dans les listes, ce qui implique que l'égalité suivante est toujours vérifiée : $N_{elts} = \sum N_i$.

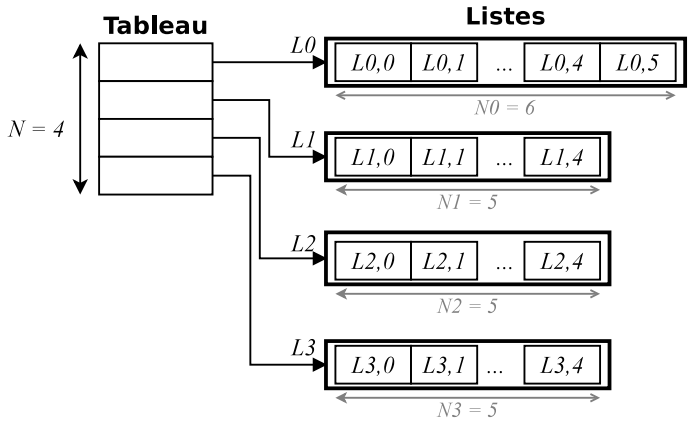


FIGURE 1 – Illustration de la structure de données.

La structure de données respecte à tout moment les contraintes suivantes :

1. **Eléments ordonnés :** Les éléments $L_{i,j}$ des listes sont triés selon leur ordre naturel (noté $\prec$). Un élément est strictement plus grand que tout élément situé avant lui dans la même liste (Eq. 1) et tout élément d'une liste qui précède sa liste (Eq. 2). Plus formellement,

$$\forall i, j \mid (0 \leq i < N) \text{ et } (0 \leq j < N_i), \begin{cases} L_{i,j} \succ L_{i,l} & \forall l \mid 0 \leq l < j \quad (1) \\ L_{i,j} \succ L_{k,l} & \forall k, l \mid (0 \leq k < i) \text{ et } (0 \leq l < N_k) \quad (2) \end{cases}$$

2. **Equilibre des longueurs de listes :** Deux listes ne peuvent avoir des longueurs qui diffèrent de plus d'une unité. De plus, les listes les plus longues sont placées en début de tableau. Cela se traduit par les longueurs de listes suivantes obtenues en fonction du nombre d'éléments stockés (N_{elts}).

$$\forall i \mid 0 \leq i < N, N_i = \begin{cases} N_{elts} \div N + 1 & \text{si } i < N_{elts} \bmod N \quad (3) \\ N_{elts} \div N & \text{sinon} \quad (4) \end{cases}$$

Une meilleure intuition du fonctionnement de la structure de données peut être obtenue en observant la Figure 2. Celle-ci donne un exemple avec des `String` où $N = 4$ et $N_{elts} = 6$. Les deux premières listes ont une taille $N_0 = N_1 = 2$ tandis que les deux dernières ont une taille $N_2 = N_3 = 1$. Les éléments sont stockés de manière ordonnée : `Abc` $\prec$ `Bidon` $\prec$ `Tonton` $\prec \cdots \prec$ `Ulysse` (ordre lexicographique des `String`).

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

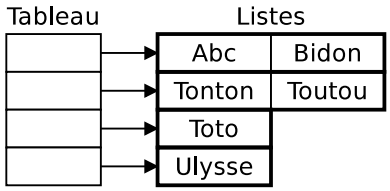
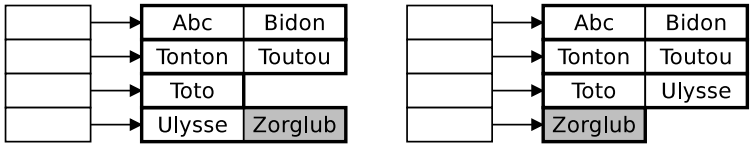


FIGURE 2 – Instance contenant des `String`.

L'objectif de la question est d'implémenter l'ajout d'un élément à la structure de données. L'ajout a pour post-condition que la structure de données (1) reste triée, (2) reste équilibrée et (3) ne comporte pas de dupliqués. Une exception doit être générée si l'élément à ajouter existe déjà. Après l'ajout d'un nouvel élément à la position dictée par l'ordre naturel, il est possible que les longueurs de listes ne soient plus équilibrées. Il s'agit alors de déplacer des éléments de façon à rétablir l'équilibre. Pour illustrer le ré-équilibrage, deux exemples d'ajouts sont illustrés aux Figures 3 et 4.

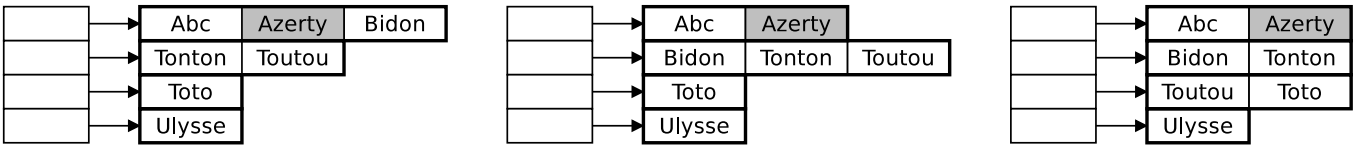
1. **Premier cas (Fig. 3) :** l'élément `Zorglub` est ajouté à la fin de la dernière liste (L_3) car il est plus grand que tous les éléments existants. La structure n'est plus équilibrée (Fig. 3a) : en effet, N_3 devrait valoir $N_{elts} \div N = 7 \div 4 = 1$ (Eq. 3). Il faut donc faire **remonter** le premier élément de L_3 vers L_2 (Fig. 3b).



(a) après l'ajout (b) après équilibrage

FIGURE 3 – Exemple d'ajout, propagation vers le haut

2. **Second cas (Fig. 4) :** l'élément `Azerty` est inséré en seconde position de L_0. La structure n'est plus équilibrée (Fig. 4a) : en effet, N_0 devrait valoir $N_{elts} \div N + 1 = 7 \div 4 = 2$ (Eq. 4). Il faut donc faire **descendre** l'élément `Bidon` de L_0 vers L_1 (Fig. 4b). La structure n'est pas encore équilibrée (N_1 devrait valoir 2). Il faut **descendre** l'élément `Toutou` de L_1 vers L_2 (Fig. 4c).



(a) après l'ajout (b) en cours d'équilibrage (c) après équilibrage

FIGURE 4 – Exemple d'ajout, propagation vers le bas

Ce qui vous est demandé :
La classe `ArrayOfListSet` met en oeuvre les principes de la structure de données décrite ci-dessus. Son implémentation est partiellement fournie ci-dessous. La variable d'instance `lists` conserve la référence vers le tableau de listes. La méthode `add` permet d'ajouter un élément à la structure de données tout en respectant les pré-conditions (ordonnée, équilibrée, absence de dupliqués). Pour vous faciliter la tâche, l'implémentation de la méthode `add` est fournie. Celle-ci repose cependant sur trois méthodes privées qu'il vous faut implémenter.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Q2a Méthode `findListIndex` détermine l'index de la liste dans laquelle l'élement `elt` doit être ajouté et retourne cet index.

Q2b Méthode `insert` insère l'élément `elt` à la bonne position dans la liste d'index `i`. Si l'élément existe déjà, la liste reste inchangée et une exception est déclenchée.

Q2c Méthode `equilibrate` vérifie si la liste d'index `i` satisfait les critères d'équilibre. Si ça n'est pas le cas, la méthode déplace des éléments dans la liste suivante ou précédente. Cette opération est répétée jusqu'à ce que la structure soit équilibrée.

```
public class ArrayOfListSet<E extends Comparable<E>> {

  private final int N;
  private final List<E> [] lists;
  private int numElts= 0;

  @SuppressWarnings(''unchecked'')
  public ArrayOfListSet(int N) {
    this.N= N;
    lists= (List<E>[]) new List[N];
    for (int i= 0; i < N; i++)
      lists[i]= new LinkedList(); // new ArrayList();
  }

  public void add(E elt) throws Exception {
    int i= findListIndex(elt);
    insert(i, elt);
    equilibrate(i);
    numElts++;
  }

  private int findListIndex(E elt) { /* Implémentation à fournir */ }
  private void insert(int i, E elt) throws Exception { /* Implémentation à fournir */ }
  private void equilibrate(int i) { /* Implémentation à fournir */ }

}
```

Note importante : Vous ne devez pas manipuler directement l'intérieur des listes ! Les listes sont des instances supportant l'interface `List`. Dans l'exemple, il s'agit d'instances de `LinkedList` mais le code pourrait être modifié de façon à utiliser `ArrayList`. Ci-dessous, quelques méthodes utiles de l'interface `List` sont rappelées. La méthode `add(E)` ajoute un élément en fin de liste. La méthode `add(int,E)` insère un élément à une position donnée. La méthode `remove(int)` enlève l'élément situé à la position donnée et le retourne.

```
public interface List<E> {

  boolean add(E elt);
  void    add(int i, E elt);
  E       remove(int i);

}
```

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2a **(méthode `findListIndex`)**

Q2b **(méthode `insert`)**

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2c **(méthode `equilibrate`)**

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2017

Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Consignes à lire impérativement !

L'examen est composé de **2 parties**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- Commencez par écrire vos **nom**, **prénom** et **section** (math, info, ...) sur chaque feuille, y compris les feuilles de brouillon.
- Laissez vos calculatrice, téléphone portable et notes de cours dans votre sac. Leur usage n'est **pas autorisé**. Pensez à éteindre votre téléphone portable !
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Utilisez pour vos réponses les **cadres** prévus à cet effet. Si davantage d'espace est nécessaire, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement où se situe le restant de la réponse.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).
- Toutes les feuilles (énoncé et brouillon) doivent être remises en fin d'examen.
- Vérifiez que vous avez répondu à toutes les questions (il y a **3 questions** dans cette partie).

Question 1 – Tri de liste chaînée (/4)

L'objectif de cette question est le tri d'une liste chaînée. La liste doit être triée en ordre croissant, en utilisant l'approche Selection sort.

La classe `LinkedList` implémente la structure de données liste chaînée. Une partie de cette classe est présentée ci-dessous. La variable d'instance `head` référence le premier noeud de la liste (tête). Les noeuds de la liste sont des instances de la classe interne `Node`. Ceux-ci comportent une référence vers la donnée stockée (`elt`) et une référence vers le noeud suivant (`next`).

```
public class LinkedList<E> {

  private Node head;

  private class Node {
    public E elt;
    public Node next;
  }

  public void sort(Comparator<E> c) { /* implementation a fournir */ }

}
```

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2017

Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Ce qui vous est demandé :

Il vous est demandé d'implémenter la méthode `sort` de la classe `LinkedList`. Cette méthode prend comme argument une instance supportant l'interface `Comparator<E>` afin de comparer entre eux les élements de la liste. L'interface `Comparator` définit une seule méthode nommée `compare(E x, E y)` qui retourne une valeur inférieure à 0 si $x < y$, une valeur supérieure à 0 si $x > y$ et 0 si $x = y$.

Note : pour des raisons de lisibilité du code, il peut être utile de créer une ou plusieurs méthodes annexes pour implémenter `sort`. L'échange des valeurs de deux noeuds de la liste pourrait par exemple être délégué à une méthode `exch` (dont l'implémentation devrait aussi être fournie dans ce cas).

Q1 **(méthode `sort`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Q1 (suite) **(méthode `sort`)**

Question 2 – Itérateur pour liste chaînée (/3)

L'objectif de cette question est de permettre le parcours d'une liste chaînée à l'aide d'un itérateur. Pour ajouter un peu de piment à la question, l'itérateur peut être muni d'un filtre qui passe certains éléments lors du parcours avec l'itérateur. Le filtre doit pouvoir être configurable. Il sera typiquement fourni à la liste sous la forme d'une expression lambda. Deux cas d'utilisation sont montrés ci-dessous.

```
LinkedList<Integer> l=
  new LinkedList<>();
l.setFilter(x -> x < 5 || x > 25);
l.add(33);
l.add(2);
l.add(17);
l.add(-1);
l.add(22);
for (Integer i: l)
  System.out.println(i);
// Résultat : 17, 22
```

```
LinkedList<String> l=
  new LinkedList<>();
l.setFilter(x -> x.charAt(0) == 'M');
l.add("Olrik");
l.add("Mortimer");
l.add("Fantasio");
l.add("Milou");
l.add("Blake");
for (String s: l)
  System.out.println(s);
// Résultat : Olrik, Fantasio, Blake
```

La classe `LinkedList` est similaire à celle utilisée dans la question précédente. Pour cette question, elle est munie d'une méthode `iterator` qui retourne une instance de l'itérateur. La classe interne `LinkedListIterator` définit le comportement de l'itérateur.

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2017

Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

```
public class LinkedList<E> {

  private Node head;

  private class Node {
    public E elt;
    public Node next;
  }

  private Filter<E> filter;

  public void setFilter(Filter<E> filter) {
    this.filter= filter;
  }

  private class LinkedListIterator implements Iterator<E> {
    /* Implémentation à fournir */
  }

  public Iterator<E> iterator() {
    return new LinkedListIterator(filter);
  }

}
```

Ce qui vous est demandé :

Q2a Définissez le type `Filter`.

Q2b Fournissez l'implémentation de la classe interne `LinkedListIterator`.

Q2a **(type `Filter`)**

.....

.....

.....

.....

.....

.....

.....

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2b **(classe `LinkedListIterator`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Question 3 – Good luck ! (/3)

Le programme suivant affiche plusieurs valeurs à la console. Il vous est demandé de fournir les valeurs affichées. Pour chacune, donnez une brève explication de votre raisonnement. Il est en particulier important d'indiquer quels types de liaisons (bindings) sont effectuées.

```
public class GoodLuck {

        public static class A {
                public static String s= "Hello";
                public int x= 10;
                public int magic() {
                        return s.length();
                }
                public int getX() {
                        setX(0);
                        return x + getX();
                }
                public void setX(int x) {
                        x= x;
                }
        }

        public static class B extends A {
                public final String s= "Bonjour";
                public int magic() {
                        return this.s.length() - s.length();
                }
                public void setX() {
                        ((A) this).setX(x);
                }
        }

        public static class C extends B {
                public static int x= 1;
                public int getX() {
                        if (x > 0)
                                return super.getX();
                        else
                                return -1;
                }
                public void setX(int x) {
                        this.x= x;
                }
        }

        (... suite page suivante ...)
```

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2017

Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

```
        public static void main(String [] args) {
                A x= new C();
                A y= new A();
                A z= new B();
                System.out.println(x.s);                                    ( Q3a )
                System.out.println(((B) x).s);                              ( Q3b )
                System.out.println(x.magic());                              ( Q3c )
                System.out.println(y.magic());                              ( Q3d )
                System.out.println(x.s == y.s);                             ( Q3e )
                z.setX(3);
                System.out.println(z.x);                                    ( Q3f )
                System.out.println(x.getX());                               ( Q3g )
                System.out.println(x.getX());                               ( Q3h )
                y.setX(100);
                System.out.println(y.x);                                    ( Q3i )
                System.out.println(y.getX());                               ( Q3j )
        }

}
```

Q3a **(x.s)**

.....

.....

Q3b **(((B) x).s)**

.....

.....

Q3b **(x.magic())**

.....

.....

Q3d **(y.magic())**

.....

.....

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2017
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q3e **(`x.s == y.s`)**

. . . . . .

. . . . . .

Q3f **(`z.x`)**

. . . . . .

. . . . . .

Q3g **(`x.getX()`)**

. . . . . .

. . . . . .

Q3h **(`x.getX()`)**

. . . . . .

. . . . . .

Q3i **(`y.x`)**

. . . . . .

. . . . . .

Q3j **(`y.getX()`)**

. . . . . .

. . . . . .