Année spéciale 2002-2003. Contrôle n°1

Structure de données :

Toutes les questions portent sur une structure de données qu'il est nécessaire de bien comprendre d'abord. Cette structure est composée de 3 tableaux parallèles (les informations liées au même indice dans les 3 tableaux sont liées). Le premier tableau contient une suite de nombres entiers quelconques non ordonnés. Au cours de l'exécution du logiciel, de nouvelles valeurs seront ajoutées et certaines valeurs disparaîtront. Pour éviter de procéder à de coûteux décalages (pour tasser les valeurs après une disparition ou pour faire de la place à une valeur quand elle arrive), nous voudrions gérer les places libres et les places occupées. Un second tableau de booléens nous indiquera, pour chaque case du premier tableau, si elle contient une valeur effectivement présente ou si elle ne contient pour l'instant rien d'utile.

Par exemple, nous pourrions avoir :

4	8	9	13	25	28	36	42	45	53
V	V	F	V	V	F	V	F	F	V

Les valeurs effectivement présentes sont : 4, 8, 13, 25, 36, 53. Les autres ont été « supprimées ». Pour gagner du temps lors d'explorations des valeurs, nous voudrions savoir où se trouve la valeur suivante, pour chacune des valeurs effectivement présentes. Pour cela, nous disposons d'un entier indiquant où est la première valeur utile (dans notre exemple en 1) d'un troisième tableau ayant, dans le même exemple, ce contenu :

4	8	9	13	25	28	36	42	45	53
2	4		5	7		10			0

Le successeur de l'élément de la case 1 se trouve dans la case 2, …le successeur de l'élément de la case 7 se trouve dans la case 10, l'élément de la case 10 n'a pas de successeur, ce qui par convention est noté 0.

De la même manière, les cases libres seront chaînées, avec un entier indiquant où est la première case libre (ici 3), et le même tableau ainsi complété :

4	8	9	13	25	28	36	42	45	53
2	4	6	5	7	8	10	9	0	0

Question 1

Ecrire un algorithme qui reçoit le tableau de booléens **`lesBooléens`**, de **`n`** éléments, qui indique combien il y a d'éléments valides. Dans l'exemple donné, il retournerait 6.

Question 2

Ecrire un algorithme qui reçoit le tableau des liens **lesLiens**, de n éléments et l'entier **premierValide** qui indique où se trouve le premier élément valide, qui indique combien il y a d'éléments valides. Dans l'exemple donné, il retournerait aussi 6.

Question 3

Indiquer quel est le résultat de cet algorithme lorsqu'il est appelé par **mystere(lesNombres, lesBooléens, lesLiens, n, premierValide)** :

```
fonction mystere(lesNombres, lesBooléens, lesLiens, n, i)retourne
entier
paramètres
   | lesNombres tableau(n) d'entiers(E)
   | lesBooléens tableau(n) de booléens (E)
   | lesLiens tableau(n) d'entiers (E)
   | (n, i) entiers (E)
début
si i<0 ou i>n alors leverException
si i=0 alors retour 0
retour lesNombres(i)+
     mystere(lesNombres, lesBooléens, lesLiens, n, lesLiens(i))
fin
```

Question 4

Ecrire un algorithme qui reçoit le tableau des nombres et le tableau des booléens, qui détermine les valeurs des deux variables indiquant les points de départ des chaînages et construit le tableau des liens.

Année spéciale 2002-2003. Corrigé du contrôle n°1

Question 1

```
fonction compte(lesBooléens, n) retourne entier
paramètres
  | lesBooléens tableau(n) de booléens (E)
  | n entier (E)
variables
  | (i, cpt) entiers
début
cpt := 0
pour i :=1 à n faire
  | si lesBooléens(i) alors cpt :=cpt+1
retour cpt
fin
```

Question 2

```
fonction compte(lesLiens, n, premierValide) retourne entier
paramètres
  | lesLiens tableau(n) d'entiers (E)
  | (n, premierValide) entiers (E)
variables
  | (i, cpt) entiers
début
cpt := 0
i := premierValide
tantque i≠0 faire
  | cpt :=cpt+1
  | i := lesLiens(i)
retour cpt
fin
```

Question 3

L'algorithme calcule la somme des éléments valides du tableau.

Question 4

```
procédure construitLiens(lesNombres, lesBooléens, n, lesLiens,
premierValide, premierLibre)
paramètres
   | lesNombres tableau(n) d'entiers(E)
   | lesBooléens tableau(n) de booléens (E)
   | lesLiens tableau(n) d'entiers (S)
   | n entier (E)
   | (premierValide, premierLibre) entiers (S)
variables
   | (ancienLibre, ancienValide, i) entiers
début
premierValide :=0
premierLibre :=0
ancienLibre :=0
ancienValide :=0
pour i := 1 à n faire
   | si lesBooléens(i) alors
   |   | si premierValide=0 alors premierValide := i
   |   | si ancienValide≠0 alors lesLiens(ancienValide) := i
   |   | ancienValide := i
   |  sinon
   |   | si premierLibre=0 alors premierLibre := i
   |   | si ancienValide≠0 alors lesLiens(ancienLibre) := i
   |   | ancienLibre :=i
lesLiens(ancienValide) :=0
lesLiens(ancienLibre) :=0
fin
```

Année spéciale 2002-2003. Contrôle du second semestre

Question 1

Ecrire un algorithme qui indique combien de fois une information donnée apparaît dans un arbre binaire. Les fonctions disponibles de la classe **`ArbreBinaire`** sont : la **`fonction estVide(unArbre) retourne booléen`**, la **`fonction sousArbreGauche(unArbre) retourne ArbreBinaire`**, la **`fonction sousArbreDroit(unArbre) retourne ArbreBinaire`**, la **`fonction valeur(unArbre) retourne Info`**, l'opération **=** entre deux objets de la classe **`Info`**.

Question 2

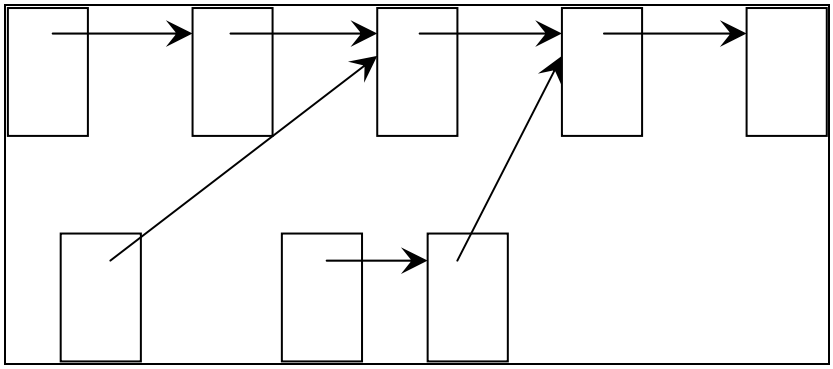
Ecrire un algorithme effectuant le même travail dans un **`ABR`**. On supposera que lors du stockage dans l'**`ABR`**, les valeurs strictement inférieures à la valeur de la racine sont stockées dans le sous arbre gauche, tandis que les autres sont stockées dans le sous arbre droit.

Question 3

Une classe **ABRD** dérive de la classe **ABR** (qui dérive elle-même de la classe **ArbreBinaire**). Cette classe ajoute aux attributs des arbres binaires un attribut supplémentaire privé **père**, de type pointeur sur **ArbreBinaire**, qui est supposé permettre de stocker, dans un **ABRD**, l'adresse de l'arbre dont notre sommet « racine » est, soit le fils gauche, soit le fils droit. La construction d'un **ABRD** se fait en déclenchant d'abord la construction d'un **ABR** et en y inscrivant les informations qu'il doit contenir, puis en appelant un algorithme de conversion qui inscrit les bonnes valeurs dans l'attribut supplémentaire **père**. Ecrire cet algorithme de conversion sous la forme d'une procédure recevant l'**ABRD** en entrée/sortie. Nous disposons des outils habituels et de la **procédure inscritPere(unABRD, unPointeur)** qui inscrit la valeur de pointeur donnée dans le champ **père** de l'**ABRD** donné.

Question 4

Nous disposons de deux listes chaînées et voudrions savoir si leurs composants sont disjoints ou non.

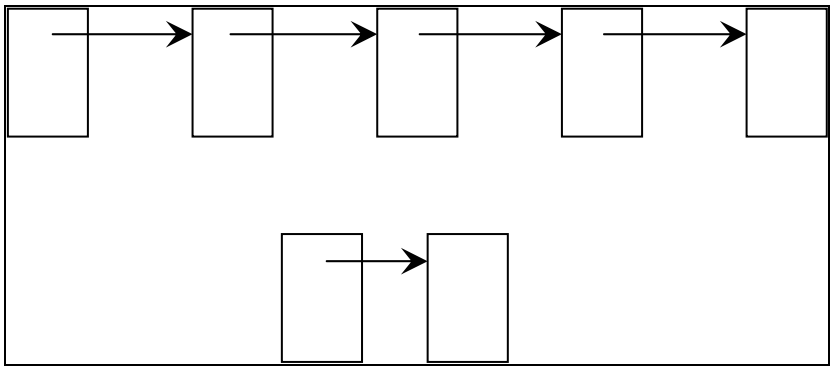


Dans cet exemple, nous avons trois listes chaînées, la première comportant 5 cellules, la seconde en comportant 4 (avec 3 cellules communes avec la première), la dernière en comportant 4 (avec 2 cellules communes avec la première et la seconde).

Deux listes peuvent parfaitement contenir les mêmes informations sans pour autant partager les mêmes cellules comme c'est le cas dans le dessin ci-dessus.

La **fonction adresse(uneListe) retourne pointeur sur Liste** permet de savoir où se trouve physiquement en mémoire la cellule de tête de la liste donnée.

Ecrire un algorithme auquel on communique deux listes, qui indique si elles partagent au moins une cellule. La réponse de cet algorithme pour n'importe quel couple des listes de notre dessin serait positive. Elle serait négative pour ces deux listes, quelles que soient les valeurs portées par les cellules :



Indiquer pourquoi la solution suivante est fausse en général et dans quels cas elle détecterait la présence de composants communs :

```
fonction composantsCommuns(l1, l2) retourne booléen
paramètres
  | (l1, l2) Listes (E)
début
si estVide(l1) ou estVide(l2) alors retour FAUX
retour adresse(l1)=adresse(l2) ou
     composantsCommuns(sousListe(l1), sousListe(l2))
fin
```

Année spéciale 2002-2003. Corrigé du contrôle du second semestre

---

Question 1

```
fonction nbOcc(unArbre, uneInfo)retourne entier
paramètres
  | unArbre ArbreBinaire (E)
  | uneInfo Info (E)
début
si estVide(unArbre) alors retour 0
si uneInfo=valeur(unArbre) alors
  | retour 1+nbOcc(sousArbreGauche(unArbre), uneInfo)+
  |   nbOcc(sousArbreDroit(unArbre), uneInfo)
retour nbOcc(sousArbreGauche(unArbre), uneInfo)+
  nbOcc(sousArbreDroit(unArbre), uneInfo)
fin
```

Question 2

```
fonction nbOcc(unArbre, uneInfo)retourne entier
paramètres
  | unArbre ABR (E)
  | uneInfo Info (E)
début
si estVide(unArbre) alors retour 0
si uneInfo > valeur(unArbre) alors
  | retour nbOcc(sousArbreDroit(unArbre), uneInfo)
si uneInfo=valeur(unArbre) alors
  | retour 1+ nbOcc(sousArbreDroit(unArbre), uneInfo)
retour nbOcc(sousArbreGauche(unArbre), uneInfo)
fin
```

Question 3

```
procédure conversion(unABRD)
paramètre
  | unABRD ABRD(E/S)
début
si estVide(unABRD) alors retour
conversion(sousArbreGauche(unABRD))
conversion(sousArbreDroit(unABRD))
si non estVide(sousArbreGauche(unABRD)) alors
  | inscritPere(sousArbreGauche(unABRD), adresse(unABRD))
si non estVide(sousArbreDroit(unABRD)) alors
  | inscritPere(sousArbreDroit(unABRD), adresse(unABRD))
inscritPere(unABRD, NULL)
fin
```

Question 4

```
fonction composantsCommuns(l1, l2) retourne booléen
paramètres
  | (l1, l2) Listes (E)
début
si estVide(l1) ou estVide(l2) alors retour FAUX
retour adresse(l1)=adresse(l2) ou
     composantsCommuns(l1, sousListe(l2)) ou
     composantsCommuns(sousListe(l1), l2)
fin
```

La solution proposée dans l'énoncé est fausse parce qu'elle ne compare pas l'adresse de la cellule de tête de chacune des listes à l'ensemble des cellules de l'autre. Elle ne détectera des éléments communs que s'ils occupent la même position dans chacune des deux listes données.

Année spéciale 2002-2003. Devoir surveillé n°1

Question 1

Voici un algorithme écrit dans la classe Liste, qui a donc le droit d'utiliser les détails d'implémentation privés de la classe. Indiquer quel est son rôle.

```
fonction mystere(uneListe) retourne booléen
paramètre
   | uneListe Liste(E)
début
si estVide(uneListe) alors retour Vrai
si uneListe.suivante=Null alors retour Vrai
si uneListe.val>uneListe.suivante->Liste.val alors retour Faux
retour mystere(uneListe.suivante->Liste)
fin
```

Pour mémoire, la classe Liste a été définie ainsi :

```
Classe Liste
Attributs privés
   | val Info
   | suivante pointeur sur Liste
```

A votre avis, que se produirait-il si nous enlevions la ligne « **`Si uneListe.suivante=Null alors retour Vrai`** » ?

Ecrire une version de ce même algorithme hors de la classe Liste. Les méthodes disponibles sont :

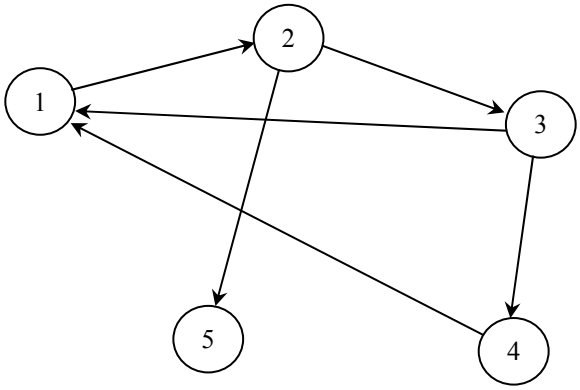
- la **`fonction valeur(uneListe) retourne Info`**, qui retourne l'information inscrite dans la cellule de tête de la liste donnée,
- la **`fonction sousListe(uneListe) retourne Liste`**, qui retourne la sous liste
- la **`fonction estVide(uneListe) retourne booléen`**.

Question 2

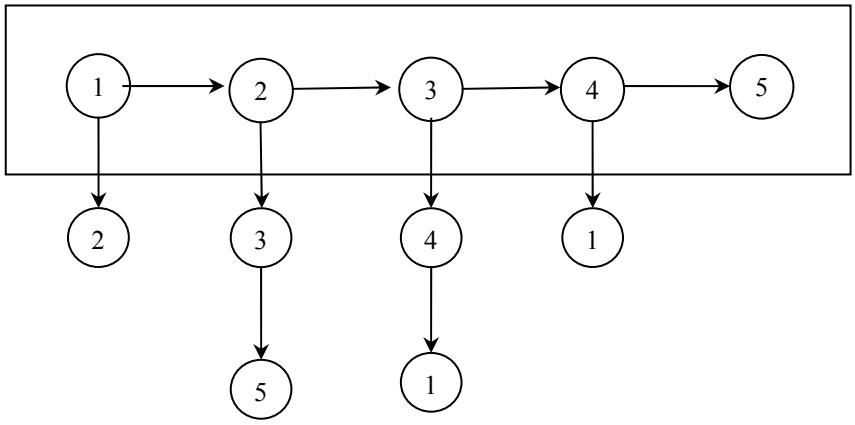
Nous voudrions maintenant pouvoir travailler dans une liste comme nous le faisons dans un tableau, en indiquant l'indice de l'élément qui nous intéresse. Ecrire une **`fonction valeur(uneListe, unePosition) retourne Info`** qui nous retourne l'information située à l'endroit indiqué dans la liste, et lève une exception si l'indice donné est fantaisiste, c'est-à-dire inférieur à 1 ou supérieur au nombre de cellules de la liste.

Question 3

Nous travaillons maintenant sur une structure de données destinée à modéliser un graphe comme celui-ci.



Notre structure de données comportera une liste chaînée des sommets du graphe, et chaque cellule de cette liste contiendra, outre le numéro du sommet (1, 2, 3, 4 ou 5), une liste des sommets destinations des arcs partant de ce sommet, ce qui nous donnera, pour l'exemple de graphe donné :



La liste encadrée sur le dessin est la liste principale. Chaque élément de cette liste contient une liste secondaire, éventuellement vide, qui indique quels sont les sommets destinations des arcs partant de là.

Question 3.1

Ecrire un algorithme qui affiche les valeurs des sommets du graphe d'où aucun arc ne part. Dans notre exemple, cet algorithme afficherait 5. Les méthodes disponibles sont :

- la **`fonction estVide(uneListe) retourne booléen`**,
- la **`fonction listeSecondaire(uneListe) retourne Liste`** qui retourne la liste secondaire attachée à la cellule de tête de la liste donnée,
- la **`procédure écrire(uneInfo)`** qui affiche l'objet de type Info qui lui est donné,
- la **`fonction sousListe(uneListe) retourne Liste`**.

Question 3.2

Ecrire une fonction à laquelle on communique deux listes, qui détermine si ces deux listes contiennent les mêmes valeurs, mais pas nécessairement dans le même ordre. La réponse de cet algorithme pour les listes {3, 4, 5} et {3, 4} serait Faux, pour {3, 4, 5} et {3, 4, 7}, ce serait Faux, pour {3, 4, 5} et {3, 4, 5}, ce serait Vrai, pour {3, 4, 5} et {5, 3, 4}, ce serait Vrai. Naturellement, cette fonction peut s'appuyer sur des sous-programmes.

Question 3.3

Nous voudrions maintenant déterminer si deux sommets du graphe se comportent de la même manière, c'est-à-dire s'ils possèdent autant d'arcs sortants et allant vers les mêmes destinations. Ecrire un algorithme qui reçoit deux sommets du graphe et indique s'ils se comportent de la même manière.

Année spéciale 2002-2003. Corrigé du DS n°1

Question 1

La fonction détermine si les éléments de la liste donnée sont ordonnés par ordre croissant au sens large. Si nous enlevions la ligne qui s'assure que la sous liste n'est pas vide, il y aurait une tentative d'accès à une zone interdite de la mémoire et probablement un arrêt brutal d'exécution…

```
fonction verifieCroissante(uneListe) retourne booléen
paramètre
  | uneListe Liste(E)
début
si estVide(uneListe) alors retour Vrai
si estVide(sousListe(uneListe)) alors retour Vrai
si valeur(uneListe)>valeur(sousListe(uneListe)) alors retour Faux
retour verifieCroissante (sousListe(uneListe))
fin
```

Question 2

```
fonction valeur(uneListe, unePosition) retourne Info
paramètres
  | uneListe Liste(E)
  | unePosition entier (E)
début
si estVide(uneListe) ou unePosition<1 alors leverException
si unePosition=1 alors retour valeur(uneListe)
retour valeur(sousListe(uneListe), unePosition-1)
fin
```

Question 3.1

```
procédure ecrirePuits(laListe)
paramètre
  | laListe Liste (E)
début
si estVide(laListe) alors retour
si estVide(listeSecondaire(laListe)) alors ecrire(valeur(laListe))
ecrirePuits(sousListe(laListe))
fin
```

Question 3.2

```
fonction semblables(liste1, liste2) retourne booléen
paramètres
  | (liste1, liste2) Listes (E)
début
si estVide(liste1) et estVide(liste2) alors retour Vrai
retour toutesPresentes(liste1, liste2) et toutesPresentes(liste2,
liste1)
fin
```

```
fonction toutesPresentes (liste1, liste2) retourne booléen
paramètres
  | (liste1, liste2) Listes (E)
début
si estVide(liste1) alors retour Vrai
retour toutesPresentes(sousListe(liste1), liste2) et
estPresente(valeur(liste1), liste2)
fin
```

```
fonction estPresente (uneInfo, liste) retourne booléen
paramètres
  | liste Liste (E)
  | uneInfo Info (E)
début
si estVide(liste) alors retour Faux
retour uneInfo=valeur(liste) ou estPresente(uneInfo,
sousListe(liste))
fin
```

Question 3.3

```
fonction semblables (sommet1, sommet2) retourne booléen
paramètres
  | (sommet1, sommet2) Listes (E)
début
retour semblables(listeSecondaire(sommet1),
listeSecondaire(sommet2))
fin
```

Année spéciale 2002-2003. Devoir surveillé n°2

Question 1

Les arbres n-aires, lorsqu'ils sont implémentés avec des sommets comportant seulement 3 pointeurs (vers le père, le fils aîné, le frère), peuvent être sérialisés (mis à plat dans un tableau) en utilisant la même technique que pour les arbres binaires, le fils aîné du sommet placé en i étant placé en 2i, le frère en 2i+1. Ecrire un algorithme qui part d'un arbre n-aire donné et emplit un tableau donné de cette manière. Le tableau doit obligatoirement être numéroté à partir de 1 et être suffisamment grand. Si le tableau se révélait mal dimensionné, cet algorithme devrait lever une exception.

Les classes disponibles sont les classes Arbre et Tableau. Il s'agit bien sûr de classes génériques, c'est à dire indépendantes des types d'informations déposées dans l'arbre ou le tableau. Les fonctionnalités disponibles pour écrire cet algorithme sont :

- la **`fonction estVide(unArbre) retourne booléen`**,
- la **`fonction premier(unTableau) retourne entier`**, qui retourne l'indice du premier élément du tableau,
- la **`fonction dernier(unTableau) retourne entier`**, qui retourne l'indice du dernier élément du tableau,
- l'**`opération ()`** de la classe Tableau, qui permet d'accéder aux éléments des instances de la classe Tableau comme on accède aux éléments des tableaux ordinaires,
- la **`fonction valeur(unArbre) retourne Info`**, qui retourne la valeur située à la racine de l'arbre,
- la **`fonction fils(unArbre) retourne Arbre`**, qui retourne le fils-aîné de l'arbre donné,
- la **`fonction frère(unArbre) retourne Arbre`**, qui retourne le frère de l'arbre donné.

Question 2

Ecrire l'algorithme de sauvegarde d'un arbre dans un fichier. Les données sont l'arbre à sauvegarder et le fichier. Les fonctionnalités disponibles sont :

- la **`fonction determineTailleTableau(unArbre) retourne entier`**, qui détermine la taille du tableau nécessaire pour y stocker les informations de l'arbre donné,
- la **`procédure écrire(unFichier, unTableau)`**, qui copie le tableau dans le fichier si celui-ci est correctement ouvert,
- les procédures habituelles pour ouvrir ou fermer un fichier.

Question 3

Ecrire une fonction qui retourne le frère aîné d'un arbre donné, c'est-à-dire l'arbre dont l'arbre donné est le frère. Les fonctionnalités disponibles sont :

- la **`fonction pere(unArbre) retourne Arbre`**, qui retourne le père de l'arbre donné,
- la **`fonction arbreVide() retourne Arbre`**, qui retourne un arbre vide.

Question 4

On vous propose cet algorithme pour la **`fonction determineTailleTableau(unArbre) retourne entier`**, qui a été utilisée dans la question 2.

```
fonction determineTailleTableau(unArbre) retourne entier
parametre
  | unArbre Arbre (E)
variable
  | resultat entier
debut
resultat := 0
parcours(unArbre, 1, resultat)
retour resultat
fin

procedure parcours(unArbre, indice, resultat)
parametres
  | unArbre Arbre (E)
  | indice entier (E)
  | resultat entier (E/S)
debut
si estVide(unArbre) alors retour
resultat := indice
parcours(fils(unArbre), 2*indice, resultat)
parcours(frere(unArbre), 2*indice+1, resultat)
fin
```

Indiquer si cet algorithme vous semble correct, par exemple après l'avoir testé sur des exemples. Si ce n'est pas le cas, proposer une modification.

Année spéciale 2002-2003. Corrigé du devoir surveillé n°2

Question 1

```
procedure serialise(unArbre, unTableau)
paramètres
   | unArbre Arbre (E)
   | unTableau Tableau (S)
début
si estVide(unArbre) alors retour
si premier(untableau)≠1 alors leverException
serialise(unArbre, unTableau, 1)
fin
```

```
procedure serialise(unArbre, unTableau, indice)
paramètres
   | unArbre Arbre (E)
   | unTableau Tableau (S)
   | indice entier (E)
début
si estVide(unArbre) alors retour
si indice>dernier(unTableau) alors leverException
unTableau(indice) := valeur(unArbre)
serialise(fils(unArbre), unTableau, 2*indice)
serialise(frere(unArbre), unTableau, 2*indice+1)
fin
```

Question 2

```
procédure sauvegarde(unArbre, unFichier)
paramètres
   | unArbre Arbre (E)
   | unFichier fichier séquentiel d'Infos (S)
début
sauvegarde(unArbre, unFichier, determineTailleTableau(unArbre))
fin
```

```
procédure sauvegarde(unArbre, unFichier, taille)
paramètres
  | unArbre Arbre (E)
  | unFichier fichier séquentiel d'Infos (S)
  | taille entier(E)
variable
  | unTableau Tableau(1, taille) d'Infos
début
ouvrir unFichier en écriture
serialise(unArbre, unTableau)
ecrire(unFichier, unTableau)
fermer(unFichier)
fin
```

Question 3

```
fonction frereAine(unArbre) retourne Arbre
paramètre
  | unArbre Arbre (E)
variable
  | unAine Arbre
début
si estVide(pere(unArbre)) alors retour pere(unArbre)
unAine := fils(pere(unArbre))
si unAine = unArbre alors retour arbreVide()
tantque frere(unAine)≠unArbre faire
  | unAine := frere(unAine)
retour unAine
fin
```

Question 4

Il faut conserver l'indice le plus grand correspondant à un sommet présent, pas le dernier rencontré, d'où la modification

```
fonction determineTailleTableau(unArbre) retourne entier
parametre
  | unArbre Arbre (E)
variable
  | resultat entier
debut
resultat := 0
si nonestVide(unArbre) alors
  | parcours(unArbre, 1, resultat)
retour resultat
fin
```

```
procedure parcours(unArbre, indice, resultat)
parametres
   | unArbre Arbre (E)
   | indice entier (E)
   | resultat entier (E/S)
debut
si estVide(unArbre) alors retour
si indice > resultat alors resultat := indice
parcours(fils(unArbre), 2*indice, resultat)
parcours(frere(unArbre), 2*indice+1, resultat)
fin
```

Année spéciale 2003-2004. Contrôle n°1

Question 1

Les algorithmes vus en exercice permettaient généralement de saisir des suites de valeurs terminées par un drapeau, par exemple des suites de lettres terminées par un point. Nous voudrions pouvoir considérer plusieurs caractères différents comme autant d'indicateurs possibles de fin de saisie. Ecrire un algorithme chargé de remplir un tableau **t** donné, de taille **n** donnée, destiné à accueillir des caractères, la saisie devant s'arrêter si un des caractères d'un second tableau **separateurs** donné, de taille **p** donnée, est rencontré.

Exemple : le tableau **separateurs** de taille 5 contient :

.	;	,	!	?

, l'utilisateur tape **il fera beau !**. Le tableau t contient en fin d'algorithme :

i	l		f	e	r	a		b	e	a	u	!

Naturellement, si la taille prévue pour le tableau accueillant le texte saisi n'était pas suffisante ou si les données n'étaient pas cohérentes, il faudrait lever une exception.

Question 2

Nous disposons maintenant de deux tableaux x et y, possédant chacun n cases contenant des valeurs entières. Ces deux tableaux contiennent les abscisses et les ordonnées de n points du plan. Un point correspond à un indice dans les deux tableaux. Ainsi, le $i^{ème}$ point du plan a pour coordonnées (x_i, y_i). Ecrire un algorithme qui détermine si les points donnés sont tous alignés ou pas, autrement dit s'il existe une droite à laquelle tous les points donnés appartiennent.

Année spéciale 2003-2004. Corrigé du contrôle n°1

Question 1

```
procedure saisie(t, n, separateurs, p)
parametres
   | t tableau(n) de caractères(S)
   | (n, p) entiers (E)
   | separateurs tableau(p) de caractères(E)
variable i entier
debut
si n ≤0 ou p ≤ 0 alors leverException
i := 1
lire(t(1))
tantque continuer(t(i), separateurs, p) faire
   | i :=i+1
   | si i>n alors leverException
   | lire (t(i))
retour
fin
```

```
fonction continuer(x, separateurs, p) retourne booléen
paramètres
   | x caractère (E)
   | separateurs tableau(p) de caractères(E)
   | p entier (E)
variable j entier
debut
si p ≤ 0 alors leverException
pour j := 1 à p faire
  | si x=separateurs(j) alors retour Faux
retour Vrai
fin
```

Question 2

```
fonction alignes(x, y, n) retourne booléen
paramètres
   | (x, y) tableaux (n) d'entiers (E)
   | n entier (E)
variables
   | i entier
   | ordinaire booléen
   | pente réel
debut
si n ≤0 alors leverException
si n ≤2 alors retour Vrai
ordinaire := x(1)≠x(2)
si ordinaire alors pente :=(y(2)-y(1))/(x(2)-x(1))
pour i :=3 à n faire
   | si ordinaire et x(i) ≠ x(2) alors
   |  | si (y(i)-y(2))/(x(i)-x(2)) ≠pente alors retour Faux
   | sinon
   |  | si ordinaire ou x(i) ≠ x(2) alors retour Faux
retour Vrai
fin
```

Année spéciale 2003-2004. Contrôle n°2

Dans l'ensemble du contrôle, nous utiliserons des tableaux non ordonnés dont les éléments pourront être valides ou non, suivant la valeur indiquée dans la case de même numéro d'un tableau de booléens associé.

Question 1

Ecrire la **`fonction placesLibres(valides, n) retourne entier`** qui calcule combien d'éléments peuvent être inscrits dans un tableau donné, à partir de son tableau de booléens associé **`valides`**, de **`n`** éléments.

Question 2

Que détermine cette fonction ?

```
fonction mystere(valides, n) retourne entier
paramètres
   | valides tableau (n) de booléens (E)
   | n entier (E)
début
retour mystere(valides, n, n)
fin
```

```
fonction mystere(valides, n, i) retourne entier
paramètres
   | valides tableau (n) de booléens (E)
   | n entier (E)
   | i entier (E)
début
si i = 0 alors retour 0
si non valides(i) alors retour i
retour mystere(valides, n, i-1)
fin
```

Question 3

Etant donnés un tableau d'entiers **t** de taille **n**, associé au tableau de booléens **valides**, lui aussi de taille **n**, écrire un algorithme qui détermine si au moins une valeur est présente plusieurs fois. Naturellement, les valeurs des cases invalides ne doivent pas être prises en compte.

Année spéciale 2003-2004. Corrigé du contrôle n°2

Question 1

```
fonction placesLibres(valides, n) retourne entier
paramètres
   | valides tableau(n) de booléens (E)
   | n entier (E)
variable
   | (resultat, i) entiers
début
pour i := 1 à n faire
   | si non valides(i) alors resultat := resultat+1
retour resultat
fin
```

Question 2

La fonction détermine le numéro du dernier emplacement libre (zéro s'il n'y en a aucun).

Question 3

```
fonction unicité(t, valides, n) retourne booléen
paramètres
   | t tableau (n) d'entiers (E)
   | valides tableau(n) de booléens (E)
   | n entier (E)
début
retour unicité(t, valides, n, n)
fin

fonction unicité(t, valides, n, i) retourne booléen
paramètres
   | t tableau (n) d'entiers (E)
   | valides tableau(n) de booléens (E)
   | n entier (E)
   | i entier (E)
début
si i=0 alors retour Vrai
si valides(i) alors
   | si estPresent(t(i), t, valides, n, i-1) alors
   |  | retour Faux
retour unicité(t, valides, n, i-1)
fin

fonction estPresent (x, t, valides, n, i) retourne booléen
paramètres
   | x entier (E)
   | t tableau (n) d'entiers (E)
   | valides tableau(n) de booléens (E)
   | n entier (E)
   | i entier (E)
début
si i=0 alors retour Faux
si valides(i) et x=t(i) alors retour Vrai
retour estPresent (x, t, valides, n, i-1)
fin
```

Année spéciale 2003-2004. Devoir surveillé 1

Question 1 (6pts)

Un fichier destiné à être visualisé sur le Web contient du texte encadré par des indications, par exemple de mise en forme, indications qui sont fournies dans des balises dont la forme générale est <balise>…</balise>. Voici un exemple d'un tel fichier :

```
<html>
	<head>
	</head>
	<body>
		<center>
			<h1>
Ceci constitue le grand titre et il sera centré et en style h1 …
			</h1>
			<h2>
Voici un titre de deuxième niveau qui sera aussi centré et en style h2 …
			</h2>
		</center>
La suite du texte ne sera plus centrée mais cadrée à gauche et en style normal …
	</body>
</html>
```

Nous voudrions vérifier à l'aide d'un automate que ce fichier respecte bien cette syntaxe, et en particulier qu'il commence par une balise <html> (elle peut être précédée d'un texte quelconque), suivie d'un texte non contrôlé, suivi d'une balise </html>. Comme dans l'exemple ci-dessus, le texte non contrôlé peut contenir d'autres balises. Dessiner le graphe de l'automate correspondant (automate sans actions).

Nous voudrions maintenant, toujours à l'aide d'un automate, vérifier que toutes les balises ouvrantes sont ensuite fermées et que les imbrications sont correctes : par exemple **<head> du texte <script> du texte </script> du texte </head>** est correct alors que : **<head> du texte <script> du texte </head> du texte </script>** ne le serait pas. Dessiner le graphe de cet automate en y plaçant les actions syntaxiques nécessaires et décrire sommairement chacune de ces actions.

Question 2 (6 pts)

Nous disposons de deux couples de tableaux t1, v1, t2, v2. Chaque couple est composé d'un tableau de données de type entier et d'un tableau de booléens chargés d'indiquer si les cases correspondantes du tableau de données sont valides ou non. Si v1(i) est vrai, cela signifie que t1(i) contient une valeur valide, autrement dit que cette case du tableau t1 n'est pas libre. Nous voudrions savoir si les deux couples ont des contenus identiques, c'est-à-dire s'ils contiennent les mêmes valeurs dans le même ordre. Il est possible que les valeurs valides ne soient pas aux mêmes endroits dans les deux tableaux.

Exemple 1 :

t1	3	5	2	6	7	1	4
v1	V	F	F	V	V	F	V
t2	3	5	2	6	7	1	4
v2	V	F	V	V	V	F	V

Ces deux couples n'ont pas les mêmes contenus (t1 contient {3, 6, 7, 4} alors que t2 contient {3, 2, 6, 7, 4}.

Exemple 2 :

t1	3	5	2	6	7	1	4
v1	V	F	F	V	V	F	V
t2	3	4	2	6	6	7	4
v2	V	F	F	F	V	V	V

Ces deux couples ont les mêmes contenus, ils contiennent tous les deux {3, 6, 7, 4}.

Les données de votre fonction seront **`t1, v1, n1, t2, v2, n2`**, n1 et n2 étant les tailles des couples de tableaux (t1, v1) et (t2, v2).

Question 3 (4 pts)

Dans le même contexte que la question 2 (couples de tableaux), indiquer ce que détermine cette fonction :

```
fonction mystere(v1, v2, n1, n2) retourne booléen
paramètres
    v1 tableau (n1) de booléens (E)
    v2 tableau (n2) de booléens (E)
    (n1, n2) entiers (E)
début
retour mystere(v1, v2, n1, n2, n1, n2)
fin
```

```
fonction mystere(v1, v2, n1, n2, i, j) retourne booléen
paramètres
    v1 tableau (n1) de booléens (E)
    v2 tableau (n2) de booléens (E)
    (n1, n2) entiers (E)
    (i, j) entiers (E)
début
si i=0 et j=0 alors retour Vrai
si i=0 ou j=0 alors retour Faux
si v1(i) et v2(j) alors retour mystere(v1, v2, n1, n2, i-1, j-1)
si v1(i) alors retour mystere(v1, v2, n1, n2, i, j-1)
retour mystere(v1, v2, n1, n2, i-1, j)
fin
```

Question 4 (4 pts)

Toujours dans le contexte des couples de tableaux, écrire un algorithme qui tasse les valeurs valides sur la gauche du couple. Si au départ, nous avons :

t1	3	5	2	6	7	1	4
v1	V	F	F	V	V	F	V

nous obtiendrons :

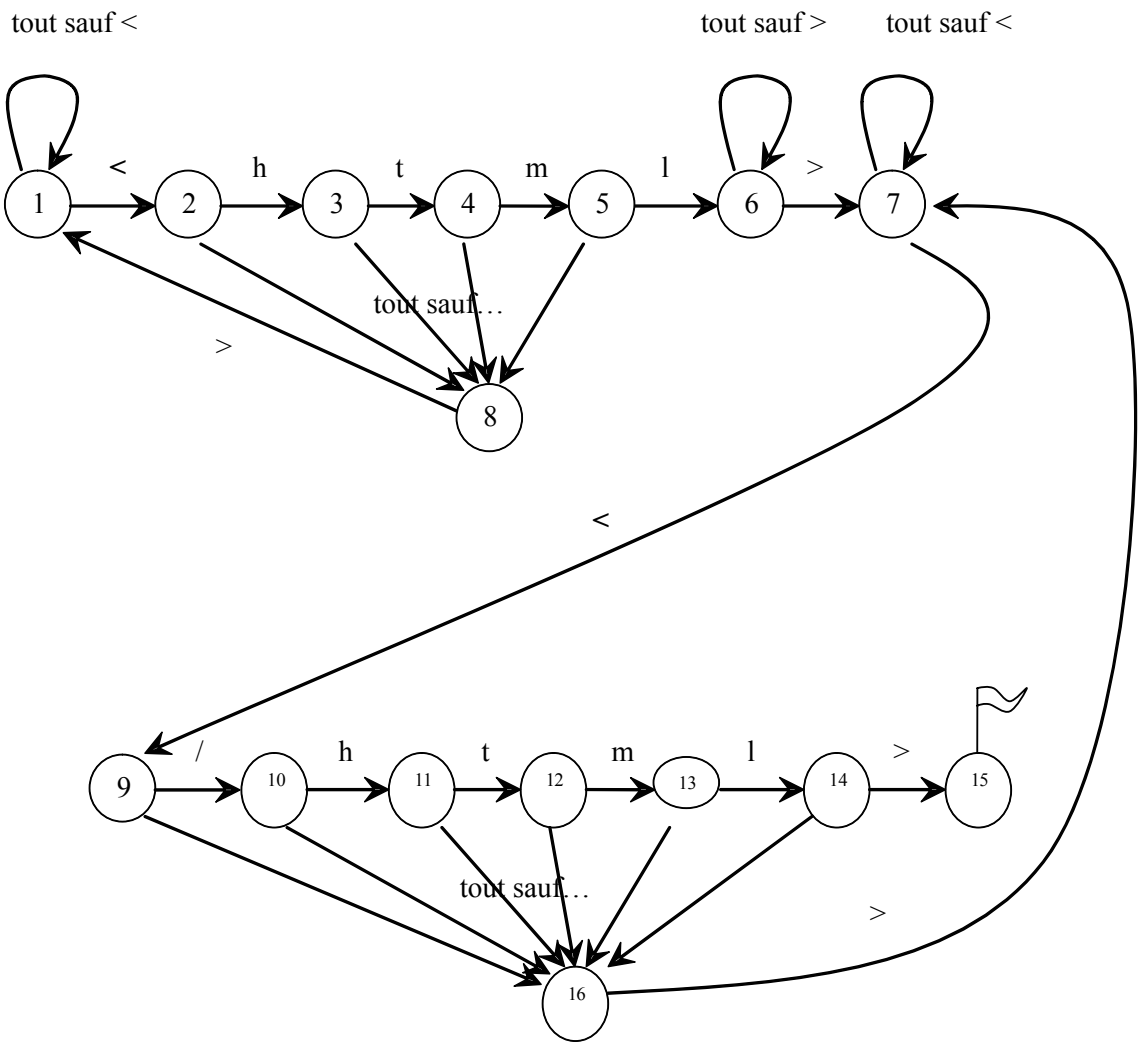
t1	3	6	7	4	7	1	4
v1	V	V	V	V	F	F	F

sachant que le contenu des cases invalides en fin de traitement n'a pas d'importance.

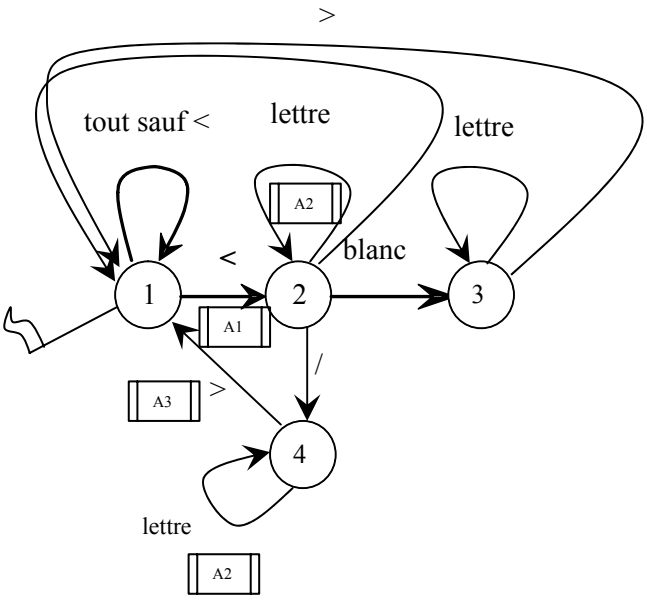
Année spéciale 2003-2004. Corrigé du devoir surveillé 1

Question 1

automate n°1 (sans action)



automate n°2 :



Rôle des actions :

En supposant que le texte en cours d'examen se situe dans le tableau **`texte`** indicé par la variable **`k`**, et que nous avons déclaré un tableau **`lesBalises`**, tableau de caractères à 2 dimensions indicé par les variables **`i`** et **`j`**, pour accueillir les libellés des balises pendant la lecture du texte :

```
A0 : i := 0
A1 : i :=i+1; j :=1; retour Vrai
A2 : lesBalises(i, j) := texte(k); j :=j+1; retour Vrai
A3 : si non compare(t, i-1, i) alors retour Faux; i :=i-2
Afinale : retour i=0
```

La fonction **`compare`** retourne **`Vrai`** si et seulement si les lignes de numéros indiqués contiennent le même libellé…

Question 2

```
fonction compare(t1, v1, n1, t2, v2, n2) retourne booléen
paramètres
   | t1 tableau(n1) d'entiers (E)
   | v1 tableau(n1) de booléens (E)
   | n1 entier (E)
   | t2 tableau(n2) d'entiers (E)
   | v2 tableau(n2) de booléens (E)
   | n2 entier (E)
variables
   | (i1, j1) entiers
début
i1 := 1
i2 := 1
tantque i1 <= n1 et i2 <= n2 faire
   | si v1(i1) et v2(i2) alors
   |  | si t1(i1)≠t2(i2)alors retour Faux
   |  | i1 := i1+1
   |  | i2 := i2+1
   | sinon
   |  | si non v1(i1) alors i1 := i1+1
   |  | si non v2(i2) alors i2 := i2+1
retour Vrai
fin
```

Question 3

La fonction indique s'il y a ou non autant de cases valides dans les deux couples.

Question 4

```
procédure tasse(t, v, n)
paramètres
   | t tableau (n) d'entiers (E/S)
   | v tableau (n) de booléens (E/S)
   | n entier (E)
variables
   | (i, k) entiers
début
k := 1
pour i := 1 à n faire
   | si v(i) et i>k alors
   |  | t(k) := t(i)
   |  | v(k) := Vrai
   |  | k := k+1
pour i := k à n faire
   | v(i) := Faux
retour
fin
```

Année spéciale 2003-2004. Contrôle n°3

Question 1

Nous disposons d'une liste chaînée construite à l'aide de la classe **`Liste`** (déjà étudiée), nous savons que les informations situées dans cette liste sont associées à des clefs numériques dont les valeurs sont comprises entre les constantes **`1`** et **`MAXCLEF`**. Par exemple, chaque information est une instance de la classe **`Piece`** et les objets de la classe **`Piece`** contiennent un attribut **`numPiece`** qui sert de clef. Nous voudrions trier cette liste par ordre croissant des clefs en utilisant la méthode suivante :

Un tableau **`aux`** numéroté de **`1`** à **`MAXCLEF`** est déclaré, ainsi qu'un tableau de booléens **`valides`** initialisé avec la valeur **`Faux`**. La liste est ensuite parcourue en déposant chacun de ses éléments dans la case du tableau dont le numéro est égal à la valeur de sa clef, le booléen associé étant basculé à **`Vrai`**. Quand la liste a été entièrement parcourue, le tableau contient toutes les informations de la liste dans les cases valides. Il ne reste plus qu'à recopier ces informations dans la liste, dans l'ordre où elles apparaissent dans le tableau. Il s'agit d'un algorithme linéaire, mais qui impose une zone de stockage en mémoire importante, si les valeurs possibles des clefs se situent dans une grande fourchette.

Ecrire un algorithme qui reçoit la liste à trier, le tableau et sa taille, algorithme qui réalise le tri par la méthode indiquée. Les méthodes dont vous disposez sont :

- la **`fonction estVide(uneListe) retourne booléen`**,
- la **`fonction sousListe(uneListe) retourne Liste`**,
- la **`fonction valeur(uneListe) retourne Info`**,
- la **`procédure inscritValeur(uneListe, uneInfo)`**,
- la **`fonction donneClef(uneInfo) retourne entier`**.

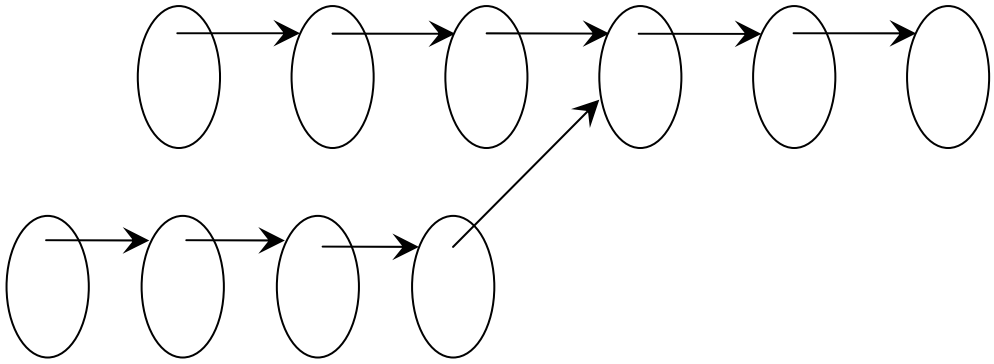
Les tableaux ayant été déclarés par le programme appelant, il est possible que leur taille ne soit pas adaptée à celle de la liste. Votre algorithme devra se montrer robuste de ce point de vue, c'est-à-dire lever une exception s'il s'en aperçoit. En outre, l'algorithme suppose que les clefs sont toutes distinctes, il devra donc lever une exception s'il s'aperçoit en cours de traitement que ce n'est pas le cas.

Question 2

Après le travail réalisé par l'algorithme de la question 1, la liste obtenue est normalement triée par ordre croissant. Ecrire un algorithme (fonction booléenne) qui vérifie que c'est bien le cas.

Question 3

Nous disposons de deux listes **`uneListe`** et **`uneAutreListe`** apparemment différentes, et nous voudrions savoir si elles ont une fin commune, comme ce serait le cas dans cette situation :



Ecrire une fonction booléenne qui reçoit deux listes est répond Vrai si et seulement si elles ont une fin commune. En plus des méthodes usuelles de la classe Liste, nous disposons de la **`fonction valeurPointeur(uneListe) retourne pointeur sur Liste`** qui retourne la valeur de pointeur inscrite dans la cellule de tête de la liste donnée, et de l'**`opération =`** entre deux pointeurs sur Listes.

Année spéciale 2003-2004. Corrigé du contrôle n°3

Question 1 (solution récursive)

```
procedure triListe(uneListe, unTableau, valides, saTaille)
paramètres
    uneListe Liste d'Infos(E/S)
    unTableau tableau(saTaille) d'Infos (E/S)
    valides tableau(saTaille) de booléens (E/S)
    saTaille entier (E)
début
copieListe(uneListe, unTableau, valides, saTaille)
copieTableau(uneListe, unTableau, valides, saTaille, 1)
fin
```

```
procedure copieListe(uneListe, unTableau, valides, saTaille)
paramètres
   | uneListe Liste d'Infos(E)
   | unTableau tableau(saTaille) d'Infos (E/S)
   | saTaille entier (E)
variable
   | indice entier
début
si estVide(uneListe) alors retour
indice := donneClef(valeur(uneListe))
si indice > saTaille alors leverException
si valides(indice) alors leverException
unTableau(indice) := valeur(uneListe)
valides(indice) := Vrai
copieListe(sousListe(uneListe), unTableau, valides, saTaille)
fin
```

```
procedure copieTableau(uneListe, unTableau, valides, saTaille,
indice)
paramètres
   | uneListe Liste d'Infos(E/S)
   | unTableau tableau(saTaille) d'Infos (E)
   | saTaille entier (E)
   | indice entier (E)
début
si indice > saTaille alors
   | si non estVide(uneListe) alors leverException
   | retour
si valides(indice) alors
   | inscritValeur(uneListe, unTableau(indice))
   | copieTableau(sousListe(uneListe), unTableau, valides, saTaille,
   |   indice+1)
sinon
   | copieTableau(uneListe, unTableau, valides, saTaille,
   |   indice+1)
fin
```

Question 2

```
fonction estCroissante(uneListe) retourne booléen
paramètre
   | uneListe Liste (E)
début
si estVide(sousListe(uneListe)) alors retour Vrai
retour
donneClef(valeur(uneListe))<donneClef(valeur(sousListe(uneListe)))
et estCroissante(sousListe(uneListe))
fin
```

Question 3 (solution « naturelle »)

Il faudra répondre Vrai si une des valeurs de pointeurs d'une des listes est présente dans l'autre.

```
fonction pointeurPresent(uneListe, unPointeur) retourne booléen
paramètres
  | uneListe Liste(E)
  | unPointeur pointeur sur Liste(E)
début
si estVide(uneListe) alors retour Faux
retour valeurPointeur(uneListe)=unPointeur ou
pointeurPresent(sousListe(uneListe), unPointeur)
fin
```

```
fonction finCommune(uneListe, uneAutreListe) retourne booléen
paramètres
  | (uneListe, uneAutreListe) Liste(E)
début
si estVide(uneListe) ou estVide(uneAutreListe) alors retour Faux
retour pointeurPresent(uneAutreListe, valeurPointeur(uneListe)) ou
finCommune(sousListe(uneListe), uneAutreListe)
fin
```

Question 3 (autre solution)

Si la fin des deux listes est commune, alors les dernières cellules des deux listes sont identiques…

```
fonction finCommune(uneListe, uneAutreListe) retourne booléen
paramètres
  | (uneListe, uneAutreListe) Liste(E)
variables
  | (liste1, liste2) Listes
début
si estVide(uneListe) ou estVide(uneAutreListe) alors retour Faux
liste1 := uneListe
liste2 := une AutreListe
tantque non estVide(sousListe(liste1)) faire
  | liste1 := sousListe(liste1)
tantque non estVide(sousListe(liste2)) faire
  | liste2 := sousListe(liste2)
retour valeurPointeur(liste1)=valeurPointeur(liste2)
fin
```

Année spéciale 2003-2004. Devoir surveillé n°2

Question 1

Des informations de la classe **`Info`** ont été déposées dans un arbre n-aire, instance de la classe Arbre qui propose les méthodes suivantes :

- La **fonction estVide(unArbre) retourne booléen**,
- La **fonction fils(unArbre) retourne Arbre**, qui retourne le premier fils de l'arbre donné,
- La **fonction frère(unArbre) retourne Arbre**, qui retourne le premier frère de l'arbre donné,
- La **fonction père(unArbre) retourne Arbre**, qui retourne le père de l'arbre donné,
- La **fonction valeur(unArbre) retourne Info**, qui retourne l'information inscrite à la racine de l'arbre donné,
- l'**opération = (uneInfo, uneAutreInfo) retourne booléen**, qui permet de comparer deux informations.

Ecrire une **fonction unicité(unArbre) retourne booléen**, qui retournera **Vrai** si aucune information n'est présente plus d'une fois dans l'arbre, **Faux** sinon.

Question 2

Les informations stockées dans notre arbre sont des instances de la classe **Info** qui dérive de la classe **ObjetV**. Cette dernière classe propose une **fonction estValide(unObjetV) retourne booléen** qui indique si l'objet en question est valide ou non et une **procédure inscritValide(unObjet, unBooléen)** qui attribue à l'objet donné le statut correspondant au booléen donné. Ecrire une **procédure inverseValide (unArbre)**, qui explore tout l'arbre qui lui est donné et inverse les validités de toutes les informations qui s'y trouvent : il rend valides celles qui ne l'étaient pas et rend invalide celles qui l'étaient.

Question 3

Un arbre binaire est finalement un cas particulier d'arbre n-aire, alors que l'inverse n'est pas vrai. Ecrire une fonction à laquelle on transmet un arbre binaire contenant des informations issues d'une classe dérivant de la classe **objetV**, qui construit et retourne un arbre n-aire ayant les mêmes informations aux mêmes endroits. Les méthodes disponibles sont les suivantes :

- la **fonction estVide(unArbreBinaire) retourne booléen**,
- la **procédure inscritValide(uneInfo, unBooléen)** déjà décrite,
- la **procédure inscritValeur(unArbre, une Info)** qui inscrit l'information donnée à la racine de l'arbre donné,
- la **procédure accrocheFils(unArbre, unAutreArbre)** qui accroche le second arbre en qualité de premier fils du premier,
- la **procédure accrocheFrere(unArbre, uneAutreArbre)** qui accroche le second arbre en qualité de frère du premier,
- les fonctions **sousArbreGauche(unArbreBinaire)** et **sousArbre-Droit(unArbreBinaire)** usuelles, les fonctions **fils(unArbre)** et **frère(unArbre)** usuelles.

On considèrera que le fait de déclarer dans un algorithme une variable locale du type **Arbre** crée automatiquement une instance de la classe Arbre et que le fait d'inscrire une valeur dedans la rend non vide. Dans le cas particulier d'un arbre binaire n'ayant pas de sous-arbre gauche mais possédant un sous-arbre droit, la technique à employer consiste à créer pour le sous-arbre gauche vide un arbre dont la racine est marquée « invalide », ce nouvel arbre ayant comme frère le résultat de la transformation en arbre du sous-arbre droit…

Année spéciale 2003-2004. Corrigé du devoir surveillé n°2

Question 1

Voici une solution simple mais mauvaise…

```
fonction unicite(unArbre) retourne booléen
paramètre unArbre Arbre (E)
début
si estVide(unArbre) alors retour Vrai
retour non estPresente(valeur(unArbre), fils(unArbre)) et non
estPresente(valeur(unArbre), frere(unArbre)) et
unicite(fils(unArbre)) et unicite(frere(unArbre))
fin

fonction estPresente(uneInfo, unArbre) retourne booléen
paramètres
   | uneInfo Info (E)
   | unArbre Arbre (E)
début
si estVide(unArbre) alors retour Faux
retour valeur(unArbre)=uneInfo ou estPresente(uneInfo,
fils(unArbre)) ou estPresente(uneInfo, frere(unArbre))
fin
```

Elle est mauvaise parce qu'elle ne détecte pas la présence éventuelle de la même information dans l'arbre fils et dans l'arbre frère de l'arbre donné…elle ne s'assure que de l'unicité de la valeur racine de l'arbre donné dans ses sous-arbres.

Question 2

```
procedure inverseValide(unArbre)
paramètre unArbre Arbre(E/S)
début
si estVide(unArbre) alors retour
inscritValide(valeur(unArbre), non estValide(valeur(unArbre)))
inverseValide(fils(unArbre))
inverseValide(frere(unArbre))
fin
```

Question 3

```
fonction transforme(unArbreBinaire) retourne Arbre
paramètre unArbreBinaire ArbreBinaire(E)
variable
  | resultat Arbre
début
si estVide(unArbreBinaire) alors
  | inscritValide(valeur(resultat), Faux)
  | retour resultat
inscritValide(valeur(resultat), Vrai)
inscritValeur(resultat, valeur(unArbreBinaire))
accrocheFils(resultat, transforme(sousArbreGauche(unArbreBinaire)))
accrocheFrere(fils(resultat),
transforme(sousArbreDroit(unArbreBinaire)))
retour resultat
fin
```

Année spéciale 2004-2005. Contrôle n°1

Question 1

Nous disposons dans un tableau **t** d'un ensemble de n valeurs strictement positives et toutes distinctes. Ces valeurs doivent être copiées dans un autre tableau **res** par la technique suivante : Le tableau **t** est traité séquentiellement, de la case n° 1 à la case n° n. La première valeur est déposée dans la case n° 1 du tableau **res**. Pour chaque valeur suivante, en commençant avec i = 1, on la compare à la valeur située dans la case i du tableau **res**. Si elle est inférieure, on s'intéresse à la case de numéro 2i du tableau **res**. Si elle est supérieure, on s'intéresse à la case 2i+1 du tableau **res**. Ce processus est réitéré jusqu'à ce que l'on trouve une case libre (valeur = 0) où notre nouvelle valeur est inscrite, et on recommence jusqu'à avoir placé toutes les valeurs du tableau **t**… Voici un exemple de tableau **t** : {6, 3, 8, 12, 1}. Le tableau **res** contiendra : {6, 3, 8, 1, 0, 0, 12}. La valeur 6 est d'abord déposée en 1. 3 étant inférieur à 6, on examine la case 2*1=2 qui est libre, on l'y installe. 8 étant supérieur à 6, on examine la case 2*1+1=3 qui est libre, on l'y installe. 12 étant supérieur à 6, on examine la case 3 qui n'est pas libre. 12 étant supérieur à 8 (actuel occupant de la case 3), on examine la case 2*3+1=7 qui est libre, on l'y installe. 1 est inférieur à 6, on examine la case 2*1=2 qui est occupée par 3. 1 est inférieur à 3, on examine donc la case 2*2=4 qui est libre, on l'y installe. Ecrire une procédure à laquelle on communique le tableau de départ et sa dimension, le tableau résultat et sa dimension, qui réalise l'inscription dans le tableau résultat des valeurs du tableau de départ en respectant notre algorithme.

Question 2

Compte tenu du mode de construction du tableau résultat décrit dans la question précédente, il est impossible d'avoir **res(i)=0** et **res(2*i)≠0** ou **res(2*i+1)≠0**. Ecrire une fonction booléenne qui reçoit le tableau résultat et sa dimension et vérifie que c'est bien le cas.

Question 3

Indiquer ce que vérifie la fonction suivante :

```
fonction mystere(t, n) retourne booléen
parametres
   | t tableau(n) d'entiers (E)
   | n entier (E)
variable
   | i entier
debut
pour i :=1 à n faire
   | si i modulo 2 = 0 et t(i) modulo 2 ≠ 0 ou
   |   i modulo 2 ≠ 0 et t(i) modulo 2 = 0 alors retour Faux
retour Vrai
fin
```

Année spéciale 2004-2005. Corrigé du contrôle n°1

Question 1

```
procedure copie(t, n, res, p)
paramètres
   | t tableau (n) d'entiers (E)
   | res tableau (p) d'entiers (S)
   | (n, p) entiers (E)
variables
   | i entier
debut
pour i := 1 à n faire
   | inscrit(res, p, 1, t(i))
fin
procedure inscrit(res, p, i, valeur)
   | res tableau (p) d'entiers (E/S)
   | (p, i, valeur) entiers (E)
debut
si i>p alors leverException
si res(i)=0 alors
   | res(i) := valeur
   | retour
si valeur > res(i) alors
   | inscrit(res, p, 2*i+1, valeur)
sinon
   | inscrit(res, p, 2*i, valeur)
retour
fin
```

Question 2

```
fonction verifie(res, p) retourne booléen
parametres
  | res tableau(p) d'entiers (E)
  | p entier (E)
variable
  | i entier
debut
pour i := 1 à n faire
  | si res(i)≠0 et res(i/2)=0 alors retour faux
retour vrai
fin
```

Question 3

La fonction vérifie que les cases du tableau ont des contenus ayant la même parité que leur indice.

Année spéciale 2004-2005. Devoir surveillé n°1

Question 1

La matrice de transitions d'un automate d'états finis est généralement stockée dans un tableau où les lignes correspondent aux états tandis que les colonnes correspondent aux événements qui provoquent des changements d'états, ce qui est le cas par exemple des caractères susceptibles d'être rencontrés lors de l'analyse d'un texte. Seuls les événements pour lesquels une transition est prévue proposent des valeurs différentes de zéro. Les matrices contiennent donc souvent un grand nombre de zéros. Pour qu'elles prennent moins de place, notamment lors d'un transfert, nous voudrions les stocker sous une forme plus compacte, en ne conservant que les valeurs non nulles et les numéros des colonnes où elles apparaissent. Par exemple, si une ligne de 10 colonnes contient la valeur 5 en colonne 6 et la valeur 8 en colonne 9, avec des zéros dans toutes les autres colonnes, nous remplaçons : {0, 0, 0, 0, 0, 5, 0, 0, 8, 0} par {6, 5, 9, 8}. Pour chaque colonne contenant une valeur non nulle, nous stockons son numéro puis la valeur qu'elle contient. Pour être en mesure de stocker en une seule ligne toutes les lignes de la matrice, nous déposerons un zéro au bout de chacune des lignes. Par exemple, si la ligne 1 devient {4, 3, 7, 7} et la ligne 2 {6, 5, 9, 8}, ces deux lignes seront codées {4, 3, 7, 7, 0, 6, 5, 9, 8, 0}. Voici un exemple de matrice avant et après compression.

Avant :

	1	**2**	**3**	**4**	**5**
1	0	0	2	0	0
2	0	2	3	0	0
3	3	0	0	4	0
4	0	0	0	0	0

Après : 3, 2, 0, 2, 2, 3, 3, 0, 1, 3, 4, 4, 0, 0

Pour être en mesure de déclarer le tableau qui recevra la matrice de transition en version compressée, il faut d'abord connaître sa taille. Ecrire un algorithme qui détermine la taille nécessaire pour le tableau à associer à une matrice donnée.

Question 2

Ecrire l'algorithme qui reçoit la matrice de transitions en entrée et le tableau résultat en sortie, qui emplit ce tableau.

Question 3

Ecrire l'algorithme qui reçoit le tableau comprimé et reconstruit la matrice de transitions pour que le moteur de l'automate puisse l'utiliser de la manière habituelle.

Question 4

Indiquer quel est le rôle de cet algorithme, supposé recevoir le tableau contenant la matrice de transition compressée :

```
fonction mystere(donnee, n, etat, colonne) retourne entier
paramètres
   | donnée tableau(n) d'entiers (E)
   | (n, etat, colonne) entiers (E)
début
retour mystere(donnee, n, etat, colonne, 1)
fin
```

```
fonction mystere(donnee, n, etat, colonne, indice) retourne entier
paramètres
   | donnée tableau(n) d'entiers (E)
   | (n, etat, colonne, indice) entiers (E)
début
si indice > n alors leverException
si etat=1 alors
   | si donnee(indice)=0 alors retour 0
   | si colonne=donnee(indice) alors retour donnee(indice+1)
   | retour mystere(donnee, n, etat, colonne, indice+2)
si donnee(indice)=0 alors
   | retour mystere(donnee, n, etat-1, colonne, indice+1)
retour mystere(donnee, n, etat, colonne, indice+2)
fin
```

Question 5

Ecrire un algorithme **récursif** qui reçoit un tableau de n éléments correspondant au format de notre tableau compressé, le nombre de lignes et de colonnes de la matrice de transition associée, qui vérifie que le contenu du tableau compressé est vraisemblable (il contient le bon nombre de lignes, les numéros d'états et de colonnes indiqués sont des numéros possibles). On ne vérifiera pas une éventuelle duplication de numéros de colonnes dans une même ligne.

Année spéciale 2004-2005. Corrigé du DS n°1

Question 1

```
fonction taille(trans, nl, nc) retourne entier
paramètres
   | trans tableau(nl, nc) d'entiers (E)
   | (nl, nc) entiers (E)
variable
   | (cpt, i, j) entiers
début
cpt := 0
pour i := 1 à nl faire
   | pour j := 1 à nc faire
   |  | si trans(i, j)≠ 0 alors cpt := cpt+2
   | cpt := cpt+1
retour cpt
fin
```

Question 2

```
procédure transforme(trans, nl, nc, resultat, p)
paramètres
   | trans tableau(nl, nc) d'entiers (E)
   | (nl, nc, p) entiers (E)
   | resultat tableau(p) d'entiers(S)
variables
   | (i, j, k) entiers
début
k := 1
pour i := 1 à nl faire
   | pour j := 1 à nc faire
   |   | si trans(i, j) ≠ 0 alors
   |   |   | resultat(k) := j
   |   |   | resultat(k+1) := trans(i, j)
   |   |   | k := k+2
   | resultat(k):=0
   | k:=k+1
retour
fin
```

Question 3

```
procédure recupere(trans, nl, nc, donnee, p)
paramètres
   | trans tableau(nl, nc) d'entiers (S)
   | (nl, nc, p) entiers (E)
   | donnee tableau(p) d'entiers(E)
variables
   | (i, j, k) entiers
début
pour i := 1 à nl faire
   | pour j := 1 à nc faire
   |   | trans(i, j) := 0
k := 1
i := 1
tantque k<=p faire
   | tantque donnee(k)≠0 faire
   |   | trans(i, donnee(k)) := donnee(k+1)
   |   | k :=k+2
   | i := i+1
   | k := k+1
retour
fin
```

Question 4

L'algorithme retourne l'état dans lequel le moteur doit aller quand il se trouve dans l'état donné et que se présente un caractère de la classe indiquée par le numéro de colonne.

Question 5

```
fonction estVraisemblable(donnee, p, nl, nc) retourne booléen
paramètres
   | donnée tableau(p) d'entiers (E)
   | (p, nl, nc) entiers (E)
début
retour estVraisemblable(donnée, p, nl, nc, 1)
fin
```

```
fonction estVraisemblable(donnee, p, nl, nc, indice) retourne
booléen
paramètres
   | donnee tableau(p) d'entiers (E)
   | (p, nl, nc) entiers (E)
   | indice entier (E)
début
si indice > p alors retour (nl=0)
si donnee(indice)=0 alors
  | retour estVraisemblable(donnee, p, nl-1, nc, indice+1)
si indice=p alors retour Faux
si donnee(indice)<1 ou donnee(indice)>nc ou donnee(indice+1)<=0 ou
donnee(indice+1)>nl alors retour Faux
retour estVraisemblable(donnee, p, nl, nc, indice+2)
fin
```

Année spéciale 2005-2006. Contrôle n°1

Question 1

Un tableau **t** de nombres contient les pixels d'une image. Bien que les images soient généralement rectangulaires, donc stockées dans des tableaux à deux dimensions, le choix qui a été fait a été de placer les lignes de cette image les unes à la suite des autres dans le tableau à une dimension **t**. En supposant que **n** est le nombre de lignes de l'image et **p** son nombre de colonnes, le point de coordonnées **(i, j)**, point sur la ligne **i** et la colonne **j**, est placé en : **(i-1)*p+j**. ***Toutes les numérotations de tableaux commencent en 1***.

Ecrire la fonction **ligne(x, n, p)** qui retourne le numéro de ligne du pixel stocké en position **x** dans le tableau **t**.

Ecrire la fonction **colonne(x, n, p)** qui retourne le numéro de colonne du pixel stocké en position **x** dans le tableau **t**.

Les opérations utiles pour réaliser ces fonctions sont d'une part la division, la division d'un entier par un entier donnant un entier, et le modulo, qui s'écrit **x modulo y** et donne le reste de la division de **x** par **y**. Ces deux fonctions devront s'assurer de la qualité de leurs données et de leurs résultats.

Question 2

Nous voudrions savoir s'il y a dans une image et dans une ligne donnée un trait horizontal d'une certaine couleur. Un trait horizontal est une suite de pixels ayant la même valeur. Ecrire une fonction booléenne à laquelle on passe le tableau **t**, un numéro de ligne **i** et une valeur de couleur **c**, qui indique si, oui ou non, un trait de cette couleur est présent dans cette ligne. Pour constituer un trait, il faut au moins deux pixels consécutifs.

Question 3

Dans le cas de la présence d'au moins un trait d'une certaine couleur dans une ligne, nous voudrions savoir où ils commencent et où ils finissent. Ecrire une procédure qui reçoit le tableau des pixels, un numéro de ligne, une couleur et emplit un tableau contenant les positions de début des traits de cette couleur et un autre tableau contenant les positions des fins de traits de cette même couleur. La procédure indiquera en plus combien de traits elle a repéré dans la ligne donnée.

Question 4

Que détermine la fonction suivante, sachant que la fonction **position(i, j, p)** renvoie la position dans le tableau à 1 dimension du pixel de coordonnées **i** et **j** ?

```
fonction mystere(t, n, p, i, c) retourne booléen
paramètres
   | t tableau(n*p) d'entiers (E)
   | (n, p, i, c) entiers (E)
debut
retour mystere(t, n, p, i, c, 1, Faux)
fin
```

```
fonction mystere(t, n, p, i, c, j, vu) retourne booléen
paramètres
   | t tableau(n*p) d'entiers (E)
   | (n, p, i, c, j) entiers (E)
   | vu booléen(E)
début
si j=p alors retour Faux
si t(position(i, j, p))=c et vu alors retour Vrai
retour mystere(t, n, p, i, c, j+1, (t(position(i, j, p))=c))
fin
```

Année spéciale 2005-2006. Corrigé du contrôle n°1

Question 1

```
fonction ligne(x, n, p) retourne entier
paramètres
  | (x, n, p) entiers (E)
variable
  | resultat entier
début
si x < 0 ou x > n*p alors leverException
si n < 0 ou p < 0 alors leverException
resultat := (x-1)/p + 1
si resultat < 1 ou resultat > n alors leverException
retour resultat
fin
```

```
fonction colonne(x, n, p) retourne entier
paramètres
  | (x, n, p) entiers (E)
variable
  | resultat entier
début
si x < 0 ou x > n*p alors leverException
si n < 0 ou p < 0 alors leverException
resultat := (x-1)modulo p + 1
si resultat < 1 ou resultat > p alors leverException
retour resultat
fin
```

Question 2

```
fonction position(i, j, p) retourne entier
paramètres
  | (i, j, p) entiers (E)
début
retour (i-1)*p+j
fin
```

```
fonction traitPresent(t, n, p, i, c) retourne booléen
paramètres
   | t tableau(n*p) d'entiers (E)
   | (n, p, i, c) entiers (E)
variable j entier
début
pour j:= 1 à p-1 faire
   | si t(position(i, j, p))=c et t(position(i, j+1, p))=c alors
   |  | retour Vrai
retour Faux
fin
```

Question 3

```
procédure traiteLigne(t, n, p, i, c, debutsTraits, finsTraits, nb)
paramètres
   | t tableau(n*p) d'entiers (E)
   | (n, p, i, c) entiers (E)
   | (debutsTraits, finsTraits) tableaux (p/3) d'entiers (S)
   | nb entier (S)
variables
   | (j, k) entiers
   | rienVu booléen
début
rienVu := Vrai
k := 1
pour j :=1 à p faire
   | si t(position(i, j, p))=c et rienVu alors
   |  | rienVu := Faux
   |  | debutsTraits(k) := j
   | sinon
   |  | si t(position(i, j, p))=c alors
   |  |  | finsTraits(k) := j
   |  | sinon
   |  |  | si non rienVu alors
   |  |  |  | k := k+1
   |  |  |  | rienVu := Vrai
si rienVu alors nb := k-1 sinon nb := k
retour
fin
```

Question 4

La fonction donnée détermine la présence d'un trait dans la ligne donnée, c'est une solution récursive à la question 2.

Année spéciale 2005-2006. Devoir surveillé

Question 1

Un texte, fourni dans un tableau de caractères, est supposé contenir l'adresse email d'une personne. Cette adresse peut commencer par un nombre quelconque d'espaces blancs, puis elle comporte soit un nom (suite de caractères ordinaires : lettres entre a et z ou entre A et Z, tiret -) soit un prénom, un point, un nom, ensuite on doit trouver le @, puis une suite de noms séparés par des points, l'ensemble pouvant être suivi d'un nombre quelconque d'espaces blancs. Par exemple : Jean-Pierre.Fournier@iut-orsay.fr, momo@wanadoo.fr, john.smith@leeds.ac.uk. Un seul point au maximum est autorisé avant le @ mais cette limitation ne se retrouve pas au-delà du @. Il n'est pas autorisé que deux points se suivent.

Donner le graphe de l'automate qui validera un tel texte.

Question 2

Indiquer quelle est la propriété des tableaux t1 et t2 que vérifie la fonction **`mystere(t1, t2, n, p)`** et donner un exemple obtenant Vrai et un exemple obtenant Faux :

```
fonction mystere(t1, t2, n, p) retourne booléen
paramètres
   | t1 tableau(n) de caractères (E)
   | t2 tableau(p) de caractères (E)
   | n, p entiers (E)
début
retour mystere(t1, t2, n, p, 1, 1)
fin
```

```
fonction mystere(t1, t2, n, p, i, j) retourne booléen
paramètres
   | t1 tableau(n) de caractères (E)
   | t2 tableau(p) de caractères (E)
   | n, p, i, j entiers (E)
début
si i>n et j>p alors retour Vrai
si i>n alors retour mystere(t2, p, j)
si j>p alors retour mystere(t1, n, i)
si t1(i)=' ' alors retour mystere(t1, t2, n, p, i+1, j)
si t2(j)=' ' alors retour mystere(t1, t2, n, p, i, j+1)
retour t1(i)=t2(j) et mystere(t1, t2, n, p, i+1, j+1)
fin
```

```
fonction mystere(t, n, i) retourne booléen
paramètre
   | t tableau(n) de caractères (E)
   | n, i entiers (E)
debut
si i>n alors retour Vrai
retour t(i)=' ' et mystere(t, n, i+1)
fin
```

Question 3

Un tableau contient des nombres entiers strictement positifs uniques. Lorsqu'on souhaite éliminer une des valeurs du tableau, plutôt que de décaler celles qui sont à sa droite, on remplace dans un premier temps cette valeur par son opposée (5 est remplacé par -5). Ceci permet, le cas échéant, de restaurer des valeurs ayant été supprimées. Lors de la seconde élimination de la même valeur, on l'efface vraiment en la remplaçant par 0.

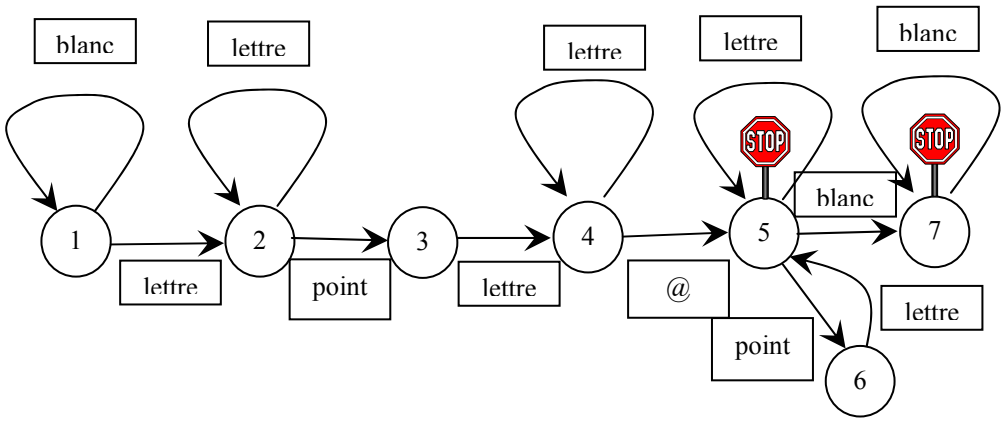
Ecrire un algorithme qui reçoit le tableau de nombres, une valeur donnée, qui élimine cette valeur, en l'inversant si elle est présente et positive, en la remplaçant par zéro si elle est présente et négative.

Ecrire un algorithme récursif qui reçoit le tableau de nombres et restaure toutes les valeurs qu'il peut restaurer.

Ecrire un algorithme qui vérifie l'unicité des valeurs dans un tableau de ce type, en prenant en compte celles qui n'ont pas été éliminées et celles qui l'ont été une seule fois.

Année spéciale 2005-2006. Corrigé du devoir surveillé

Question 1



Question 2

La fonction vérifie que les deux tableaux contiennent le même texte (la même suite de caractères dans le même ordre) en ignorant les espaces blancs. Le couple { bonjour à tous ! }, { b on jouràtous} obtient Vrai, le couple { que c'est mystérieux !}, { qu c'est mystérieux !} obtient Faux.

Question 3

```
procédure elimine(t, n, valeur)
paramètres
   | t tableau(n) d'entiers (E/S)
   | n, valeur entiers (E)
variable
   | i entier
début
pour i := 1 à n faire
   | si t(i) = -valeur alors t(i) := 0
   | si t(i) = valeur alors t(i) := -valeur
retour
fin
```

```
procédure restaure(t, n)
paramètres
   | t tableau(n) d'entiers(E/S)
   | n entier (E)
début
restaure(t, n, 1)
fin
```

```
procédure restaure(t, n, i)
paramètres
   | t tableau(n) d'entiers(E/S)
   | n, i entiers (E)
début
si i>n alors retour
si t(i) < 0 alors t(i) := -t(i)
restaure(t, n, i+1)
fin
```

```
fonction unicite(t, n) retourne booléen
paramètres
   | t tableau(n) d'entiers (E)
   | n entier (E)
début
pour i :=1 à n-1 faire
   | si estPresente(t(i), t, n, i+1) alors retour Faux
retour Vrai
fin
```

```
fonction estPresente(valeur, t, n, position) retourne booléen
paramètres
   | valeur, n , position entiers (E)
   | t tableau(n) d'entiers (E)
debut
si position > n alors retour Faux
retour abs(t(position))=valeur ou
     estPresente(valeur, t, n, position+1)
fin
```