Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Rue Royale, 67. 1000 Bruxelles
02/219.15.46 – esi@he2b.be

Algorithmique
Exercices avec solutions

2019

Bachelor en Informatique
DEV2

M. Codutti (MCD), H. Delannoy (HDE),
S. Drobisz (SDR), A. Paquot (APA) & N. Richard (NRI)

Document produit avec LaTeX.
Version du 3 février 2019.

Table des matières

Chapitre

1

Les tableaux à 2 dimensions

Exercice 1

Case nulle ?

Écrire un algorithme qui reçoit un tableau d'entiers (à n lignes et m colonnes) ainsi que les coordonnées d'une case (ligne, colonne) et qui retourne un booléen indiquant si la case désignée contient ou pas la valeur nulle.

```
algorithm estNul(tab: array of n × m integers, lg, col: integers) → boolean
```

Solution.

```
algorithm estNul(tab: array of n × m integers, lg, col: integers) → boolean
|  return tab[lg][col]=0
end
```

Exercice 2

Assigner une case

Écrire un algorithme qui reçoit un tableau d'entiers (à n lignes et m colonnes) ainsi que les coordonnées d'une case (ligne, colonne) et une valeur entière. L'algorithme met la valeur donnée dans la case indiquée pour autant que la case contienne actuellement la valeur nulle. Dans le cas contraire, l'algorithme ne fait rien.

```
algorithm assigner(tab ↕ : array of n × m integers, lg ↓ , col ↓ , val ↓ : integers)
```

Exercice 3

Un bord du tableau

Écrire un algorithme qui reçoit un tableau d'entiers (à n lignes et m colonnes) ainsi que les coordonnées d'une case (ligne, colonne). L'algorithme doit indiquer si la case donnée est ou non sur un **bord** du tableau.

```
algorithm estBord(tab: array of n × m integers, lg, col: integers) → boolean
```

Exercice 4

Un coin du tableau

Écrire un algorithme qui reçoit un tableau d'entiers (à n lignes et m colonnes) ainsi que les coordonnées d'une case (ligne, colonne). L'algorithme doit indiquer si la case donnée est ou non sur un des 4 **coins** du tableau.

```
algorithm estCoin(tab: array of n × m integers, lg, col: integers) → boolean
```

Exercice 5

Affichage

Écrire un algorithme qui affiche tous les éléments d'un tableau (à n lignes et m colonnes) ligne par ligne.

Écrivez un autre algorithme qui affiche cette fois les éléments colonne par colonne

Exercice 6 — Cases adjacentes

Écrire un algorithme qui reçoit un tableau d'entiers (à n lignes et m colonnes) ainsi que les coordonnées d'une case (ligne, colonne) et *affiche* les coordonnées des cases *adjacentes*.

Exercice 7 — Les nuls

Écrire un algorithme qui reçoit un tableau (n x m) d'entiers et qui retourne la proportion d'éléments nuls dans ce tableau.

Exercice 8 — Le tableau de cotes

Soit un tableau à n lignes et m colonnes d'entiers où une ligne représente les notes sur 20 d'un étudiant et les colonnes toutes les notes d'un cours.

Écrire un algorithme recevant ce tableau en paramètre et retournant le pourcentage d'étudiants ayant obtenu une moyenne supérieure à 50%.

Exercice 9 — Le triangle de Pascal

Le triangle de Pascal est construit de la façon suivante :

- ▷ la ligne initiale contient un seul élément de valeur 1 ;
- ▷ chaque ligne possède un élément de plus que la précédente ;
- ▷ chaque ligne commence et se termine par 1 ;
- ▷ pour calculer un nombre d'une autre case du tableau, on additionne le nombre situé dans la case située juste au-dessus avec celui dans la case à la gauche de la précédente.

Écrire un algorithme qui reçoit en paramètre un entier n, et qui renvoie un tableau contenant les $n + 1$ premières lignes du triangle de Pascal (indicées de 0 à n).

N.B. : le « triangle » sera bien entendu renvoyé dans un tableau carré (ce qui ne sera forcément le cas en Java). Quid des cases non occupées ?

Par exemple, pour n qui vaut 5, on aura le tableau ci-contre.

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

Exercice 10 — Tous positifs

Écrire un algorithme qui reçoit un tableau (n x m) d'entiers et qui vérifie si tous les nombres qu'il contient sont strictement positifs. Bien sûr, on veillera à éviter tout travail inutile ; la rencontre d'un nombre négatif ou nul doit arrêter l'algorithme.

Exercice 11 — Toute une ligne de valeurs non nulles ?

Écrire un algorithme qui reçoit un tableau d'entiers (à n lignes et m colonnes) ainsi qu'un numéro de ligne et qui retourne un booléen indiquant si la ligne donnée du tableau ne contient que des valeurs non nulles.

```
algorithm lignePleine(tab: array of n × m integers, lg: integer) → boolean
```

Faites de même pour une colonne.

Exercice 12 — Le carré magique

Un carré magique est un tableau d'entiers carré (c'est-à-dire possédant autant de lignes que de colonnes) ayant la propriété suivante : si on additionne les éléments d'une quelconque de ses lignes, de ses colonnes ou de ses deux diagonales, on obtient à chaque fois le même résultat.

Écrire un algorithme recevant en paramètres le tableau (n x n) d'entiers représentant le carré et renvoyant une valeur booléenne indiquant si c'est un carré magique ou pas.

Exercice 13 — Lignes et colonnes

Écrire un algorithme qui reçoit un tableau d'entiers à 2 dimensions en paramètre et qui retourne un booléen indiquant si ce tableau possède 2 lignes ou 2 colonnes identiques.

Dans l'affirmative, cet algorithme renverra également en paramètres les informations suivantes :

- ▷ les indices des lignes ou colonnes identiques
- ▷ un caractère valant 'L' ou 'C' selon qu'il s'agit de lignes ou de colonnes

Dans la négative, les valeurs de ces paramètres seront indéterminées ou quelconques, elles ne seront de toute façon pas utilisées par l'algorithme appelant.

Exercice 14 — Le contour du tableau

On donne un tableau d'entiers `tabEnt` à n lignes et m colonnes. Écrire un algorithme retournant la somme de tous les éléments *impairs* situés sur le bord du tableau.

Exemple : pour le tableau suivant, l'algorithme doit renvoyer 32

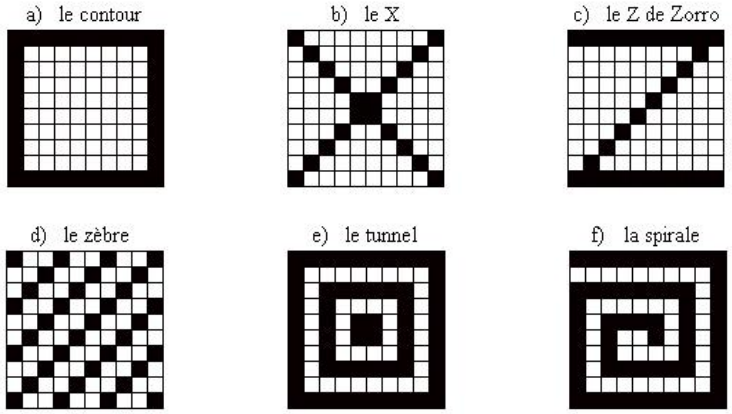
3	4	6	11
2	21	7	9
1	5	12	3

Et pour le suivant, l'algorithme doit renvoyer 6

4	1	2	8	5

Exercice 15 — À vos pinceaux !

On possède un tableau à n lignes et n colonnes dont les éléments de type Couleur valent NOIR ou BLANC. On suppose que le tableau est initialisé à BLANC au départ. Écrire un algorithme qui *noircit* les cases de ce tableau comme le suggèrent les dessins suivants (les exemples sont donnés pour un tableau 10 x 10 mais les algorithmes doivent fonctionner quelle que soit la taille du tableau).



Notes

- ▷ Le zèbre doit toujours présenter des lignes obliques et parallèles, quelle que soit la taille.
- ▷ La spirale est un véritable défi et vous est donné comme exercice facultatif. Ne le faites pas si vous êtes en retard.

Exercice 16 — Exercices sur la complexité

Quelle est la complexité

a) d'un algorithme de parcours d'un tableau n x n ?

b) des algorithmes que vous avez écrits pour les exercices : "Les nuls", "Tous positifs", "Le carré magique" et "Le contour d'un tableau" ?

c) des algorithmes que vous avez écrits pour résoudre les exercices du pinceau ?

Chapitre

2

L'orienté objet

Chapitre

3

La liste

Exercice 1 — Manipulation d'une liste

Écrire un algorithme qui crée la liste suivante :

0. 494
1. 209
2. 425

affiche sa taille, demande si la valeur 425 est présente, supprime la valeur 209 puis insère la valeur 101 en tête de liste.

Exercice 2 — Liste des premiers entiers

Écrire un algorithme qui reçoit un entier n en paramètre et retourne la liste contenant les entiers de 1 à n dans l'ordre décroissant. On peut supposer que n est strictement positif.

Exercice 3 — Somme d'une liste

Écrire un algorithme qui calcule la somme des éléments d'une liste d'entiers.

Exercice 4 — Anniversaires

Écrire un algorithme qui reçoit une liste de structure Personne (nom + prénom + date de naissance) et retourne la liste de ceux qui sont nés durant un mois passé en paramètre (donné sous la forme d'un entier entre 1 et 12).

Exercice 5 — Concaténation de deux listes

Écrire un algorithme qui reçoit 2 listes et ajoute à la suite de la première les éléments de la seconde ; la seconde liste n'est pas modifiée par cette opération.

Exercice 6 — Le nettoyage

Écrire un algorithme qui reçoit une liste de chaines en paramètre et supprime de cette liste tous les éléments de valeur donnée en paramètre. L'algorithme retournera le nombre de suppressions effectuées.

Exercice 7 — Les extrêmes

Écrire un algorithme qui supprime le minimum et le maximum des éléments d'une liste d'entiers. On peut supposer que le maximum et le minimum sont uniques.

Exercice 8 — Fusion de deux listes

Soit deux listes **triées** d'entiers (redondances possibles). Écrire un algorithme qui les fusionne. Le résultat est une liste encore triée contenant tous les entiers des deux listes de départ (qu'on laisse inchangées).

Exemple : Si les 2 listes sont (1, 3, 7, 7) et (3, 9), le résultat est (1, 3, 3, 7, 7, 9).

Exercice 9 — Éliminer les doublons d'une liste

Soit une liste **triée** d'entiers avec de possibles redondances. Écrire un algorithme qui enlève les redondances de la liste.

Exemple : Si la liste est (1, 3, 3, 7, 8, 8, 8), le résultat est (1, 3, 7, 8).

a) Faites l'exercice en créant une **nouvelle liste** (la liste de départ reste inchangée)

b) Refaites l'exercice en **modifiant** la liste de départ (pas de nouvelle liste)

Exercice 10 — Rendez-vous

Soit la structure RendezVous composée d'une date (cf. la structure Date du cours de DEV1) et d'un motif de rencontre.

```
structure RendezVous
    date: Date
    motif: string
end
```

Écrire un algorithme qui reçoit une liste de rendez-vous et la met à jour en supprimant tous ceux qui sont désormais passés.

Pour résoudre cet exercice, vous pouvez utiliser sans l'écrire un algorithme aujourdhui() qui retourne la date du jour.

Chapitre

4

Les traitements de rupture

Exercice 1

La chasse au gaspi [rupture de niveau 1]

À l'ÉSI, les quantités de feuilles imprimées et photocopiées par les professeurs et les étudiants sont enregistrées à des fins de traitement. Le service technique désirant facturer les « exagérations », vous fournit une liste de toutes les impressions effectuées depuis le début de l'année. Cette liste présente la structure d'enregistrement Job suivante et est ordonnée alphabétiquement **en majeur** sur le champ login :

```
structure Job
    login: string
    date: date
    nombre: integer
end
```

Écrire un algorithme permettant d'afficher une ligne par utilisateur dont le nombre total de feuilles imprimées dépasse une valeur limite entrée en paramètre. Cette ligne contiendra le login et le nombre.

Exercice 2

Compter le nombre d'étudiants par option

Reprenons l'exemple donné pour la rupture de niveau 2 (RuptureNiveau2, page **??**). Que faut-il ajouter à l'algorithme pour qu'il affiche également le nombre total d'étudiants par option ?

Exercice 3

Compter les étudiants [rupture de niveau 2]

Supposons que la structure Etudiant contienne également un champ indiquant dans quel bloc se trouve l'étudiant (1, 2 ou 3). On voudrait un algorithme qui reçoit une liste d'étudiants et calcule le nombre d'étudiants dans chaque section et, par section, dans chaque bloc.

L'affichage ressemblera à :

```
Gestion
    bloc 1 : 130 étudiants
    bloc 2 : 42 étudiants
    bloc 3 : 16 étudiants
    TOTAL  : 188 étudiants
Industriel
    bloc 1 : 32 étudiants
    bloc 2 : 14 étudiants
    bloc 3 : 8 étudiants
    TOTAL  : 54 étudiants
Réseau
    bloc 1 : 82 étudiants
    bloc 2 : 31 étudiants
    bloc 3 : 13 étudiants
    TOTAL  : 126 étudiants
```

a) Quel doit-être le tri de la liste pour pouvoir résoudre cet exercice avec un algorithme de rupture ?

b) Écrire cet algorithme.

Exercice 4 Les fanas d'info [rupture de niveau 2]

Une grande société d'informatique a organisé durant les douze derniers mois une multitude de concours ouverts aux membres de clubs d'informatique. Elle souhaiterait récompenser le club qui aura été le plus « méritant » durant cette période au point de vue de la participation des membres mineurs. Chaque résultat individuel des participants (y compris des majeurs) est repris dans une liste dont les éléments sont de type Participant.

```
structure Participant
    nom: string                          // nom et prénom du participant
    âge: integer                         // âge du participant au moment du concours
    référence: string                    // référence du club auquel appartient ce participant
    numéro: integer                      // numéro du concours auquel il a participé
    résultat: integer                    // résultat obtenu lors de ce concours (sur 100)
end
```

Sachant que la liste est ordonnée **en majeur sur le champ référence et en mineur sur le champ nom**, on demande d'écrire l'algorithme qui affiche les informations suivantes :

pour chaque club :

- ▷ sa référence
- ▷ pour chaque membre mineur de ce club :
  - ▷ son nom et prénom
  - ▷ la cote moyenne sur 100 des concours auquel ce membre a participé
- ▷ le nombre total de participations des membres mineurs

N.B. : un membre mineur qui s'est inscrit à un concours = une participation. Un club qui n'aura eu aucun membre mineur participant figurera quand même dans le résultat avec la mention « Pas de participation de membre mineur ». Par contre, un club dont aucun membre n'a participé au moindre concours ne sera pas affiché.

À la fin, on affichera la référence du meilleur club, à savoir celui qui a eu la plus haute cote moyenne de membres mineurs (simplifions on ne gérant pas les possibles ex-æquo).

Exercice 5 Éliminer les doublons d'une liste.

Soit une liste ordonnée d'entiers avec des possibles redondances. Écrire un algorithme qui enlève les redondances de la liste. On vous demande de créer une nouvelle liste (la liste de départ reste inchangée).

Exemple : si la liste est $(1, 3, 3, 7, 8, 8, 8)$ le résultat sera $(1, 3, 7, 8)$.

Exercice 6 Une suite logique

Voici une petite suite logique :

```
1
1 1
2 1
1 2 1 1
1 1 1 2 2 1
3 1 2 2 1 1
1 3 1 1 2 2 2 1
1 1 1 3 2 1 3 2 1 1
3 1 1 3 1 2 1 1 1 3 1 2 2 1
...
```

a) Comprendre la logique derrière cette suite et écrire la ligne suivante.

b) Écrire un algorithme qui reçoit une ligne (sous forme d'une liste d'entiers) et retourne la ligne suivante (sous forme d'une autre liste d'entiers). Votre première tâche sera probablement de comprendre ce que vient faire cet exercice dans le chapitre des ruptures puisque la liste n'est pas triée.

c) Écrire l'algorithme qui reçoit N (un entier) et affiche les N premières lignes de cette suite logique.

Exercice 7

Alternative à la rupture

Reprenons l'exemple donné pour la rupture de niveau 1 (RuptureNiveau1, page **??**). Supposons que la liste ne soit **pas** triée sur l'option. Écrivez l'algorithme qui calcule le nombre d'étudiants par option en un seul parcours de la liste (vous devrez utiliser trois compteurs distincts).

Chapitre

5

Représentation des données

Exercice 1

La course à la case 64 à 4 joueurs

Commençons par un petit jeu très simple de course avec un dé, dont voici les règles.

> « Ce jeu se joue à 4 joueurs qui doivent parcourir un chemin de 64 cases. Ils commencent tous sur la case 1 et jouent à tour de rôle (en commençant par le premier joueur). À son tour, le joueur lance un dé à 6 faces et avance du nombre de cases indiqué par le dé. Le premier joueur à atteindre ou dépasser la case 64 a gagné. Seule contrainte, un joueur ne peut pas terminer son tour sur une case occupée. Si c'est le cas, il avance jusqu'à la case libre suivante. »

Voici 3 propositions de représentation de données. On vous demande pour chaque proposition de vérifier, sans écrire l'algorithme, si elle permet la programmation du jeu. On vous conseille vivement de « dessiner » [1] les propositions pour mieux les comprendre.

1. Un tableau de 64 entiers. La case k contient i si le joueur i s'y trouve ou 0 si la case est libre. Mais aussi un entier joueurCourant donnant le numéro du joueur courant.
2. Un tableau de 4 entiers. La case i contient la position du joueur i. Mais aussi un entier joueurCourant donnant le numéro du joueur courant.
3. On combine les deux premières propositions (on a donc deux tableaux).

Après ces vérifications vous choisirez une des représentations pour écrire la solution (non OO à ce stade) du jeu. Pensez à découper votre solution.

Exercice 2

La course à la case 64 à n joueurs

Modifiez l'exercice précédent afin que le jeu puisse se jouer à n joueurs, où n est un entier supérieur ou égal à 2, choisi au début du jeu.

Exercice 3

La course à la case 64 à n joueurs - variantes

Reprenons la course à la case 64 de l'exercice précédent. Voici quelques propositions de modification des règles. Pour chaque proposition, indiquez si la représentation choisie dans l'exercice précédent est toujours valable et pertinente.

1. Si un joueur arrive sur une case occupée, le joueur qui s'y trouvait retourne à la première case.
2. Si un joueur termine sa course sur une case qui est un multiple de 5, il rejoue directement.
3. Un joueur rejoue directement s'il termine sa course sur les cases 1, 2, 7, 11, 17, 31, 42 ou 53.

1. Par là, on veut dire : imaginer une situation de jeu (positions des joueurs sur le chemin par exemple) et voir quelles valeurs doivent avoir les variables introduites dans la représentation pour correspondre à cette situation de jeu.

Exercice 4

Un jeu de poursuite

Deux joueurs A et B se poursuivent sur un circuit de 50 cases. Au départ, A se trouve sur la case 1 et B est placé sur la case 26. C'est A qui commence. Chaque joueur joue à son tour en lançant un dé dont la valeur donne le nombre de cases duquel il doit avancer sur le jeu. Lorsqu'un joueur arrive sur la case 50 et qu'il doit encore avancer, il continue son parcours à partir de la case 1. Le jeu se termine lorsqu'un joueur rattrape ou dépasse l'autre.

Écrire un algorithme (non OO pour le moment) de simulation de ce jeu qui se terminera par l'affichage du vainqueur ainsi que le nombre de tours complets parcourus par ce vainqueur.

Il est important de bien découper votre algorithme. On vous suggère d'écrire les algorithmes suivant :

- ▷ un algorithme initialiser() qui initialise le jeu (placement des joueurs...) ;
- ▷ un algorithme « jouerCoup » qui joue pour un joueur et indique s'il a rattrapé l'autre joueur ;
- ▷ un algorithme « joueurSuivant » qui permet de passer au joueur suivant.

À nouveau, on vous fait plusieurs propositions pour la représentation de l'état du jeu. On vous demande pour chacune d'elles de vérifier, sans écrire les méthodes de la classe, si elles permettent la programmation du jeu. Après ces vérifications vous choisirez une des représentations pour écrire la classe complète.

1. Dans cette proposition, nous avons deux variables.
   - ▷ circuit : un tableau de 1 à 50 chaines de caractères. Les chaines de caractères représenteront la position des joueurs (au départ, "A" en 1 et "B" en 26, " " dans les autres positions).
   - ▷ joueurCourant : un entier donnant la position du joueur courant.
2. Cette proposition introduit une structure Joueur et le nombre de tours.
   - ▷ circuit : un tableau de 1 à 50 éléments Joueur, une structure.

     ```
     structure Joueur
     |  nom: string          // Le nom du joueur à cette position ("A", "B" ou " " si la case est vide)
     |  nbTours: integer     // Nb de tours qu'a fait le joueur qui est à cette position (0 si case vide)
     end
     ```

     Un joueur a fait un tour complet quand il est de nouveau sur sa position de départ ou la dépasse.
   - ▷ joueurCourant : un entier donnant la position du joueur courant.
3. Dans cette proposition, le tableau change de signification.
   - ▷ circuit : un tableau de 2 éléments Joueur, une structure différente.

     ```
     structure Joueur
     |  position: integer    // Donne la position du joueur sur le circuit (entier entre 1 et 50)
     |  nbTours: integer                          // Le nombre de tours qu'a fait ce joueur
     end
     ```
   - ▷ joueurCourant : un entier donnant la position du joueur courant.
4. Cette proposition est identique à la précédente sauf sur un point :
   - ▷ On ne retient plus le nombre de tours effectués mais simplement le nombre de cases parcourues. Par exemple, si un joueur a fait exactement deux tours complets, le nombre de cases parcourues sera de 100.

     ```
     structure Joueur
     |  position: integer    // Donne la position du joueur sur le circuit (entier entre 1 et 50)
     |  nbCasesParcourues: integer  // Nb de cases parcourues par le joueur depuis le départ
     end
     ```

Exercice 5

Un jeu de poursuite - variante

Dans cette variante, chaque case contient une valeur vrai ou faux indiquant si le joueur pourra rejouer. Si la case sur laquelle tombe le joueur contient la valeur vrai il avance encore une fois du même nombre de cases (et de même s'il tombe encore sur vrai).

Qu'est-ce que cela change au niveau des données ? Modifiez votre solution en conséquence.

Pour adapter le code, il faudra adapter les paramètres fournis à l'algorithme d'initialisation et à l'algorithme qui joue un coup. Nous vous conseillons également de ne pas modifier l'algorithme jouerCoup mais d'introduire un nouvel algorithme (par ex. jouerTour) qui y fait appel plusieurs fois si nécessaire.

Exercice 6

Le Jeu du Millionnaire

Un questionnaire de quinze questions à choix multiples de difficulté croissante est soumis à un candidat. Quatre possibilités de réponses (dont une seule est correcte) sont proposées à chaque fois. Au plus le candidat avance dans les bonnes réponses, au plus son gain est grand. S'il répond correctement aux quinze questions, il empoche la somme rondelette de 500.000 €.

Par contre, si le candidat donne une mauvaise réponse, il risque de perdre une partie du gain déjà acquis. Cependant, certains montants intermédiaires constituent des paliers, c'est-à-dire une somme acquise que le candidat est sûr d'empocher, quoiqu'il arrive dans la suite du jeu.

À chaque question, le candidat a donc trois possibilités :

- il donne la réponse correcte : dans ce cas il augmente son gain, et peut passer à la question suivante
- il ne connait pas la réponse, et choisit de s'abstenir : dans ce cas, le jeu s'arrête et le candidat empoche le gain acquis à la question précédente
- il donne une réponse incorrecte : le jeu s'arrête également, mais le candidat ne recevra que le montant du dernier palier qu'il a atteint et réussi lors de son parcours. En particulier, si le candidat se trompe avant d'avoir atteint le premier palier, il ne gagne pas un seul euro !

1	25 €	faux
2	50 €	faux
3	125 €	faux
4	250 €	faux
5	500 €	vrai
6	1000 €	faux
7	2000 €	faux
8	3750 €	faux
9	7500 €	faux
10	12500 €	vrai
11	25000 €	faux
12	50000 €	faux
13	100000 €	vrai
14	250000 €	faux
15	500000 €	vrai

Exemple : Le tableau ci-contre contient les gains associés à chaque question et une indication booléenne mise à vrai lorsque la question constitue un palier. Un concurrent qui se trompe à la question 3 ne gagnera rien ; un concurrent qui se trompe à la question 6 gagnera 500 € (palier de la question 5) et de même s'il se trompe à la question 10 ; un concurrent qui se trompe à la question 13 gagnera 12500 € (palier de la question 10) ; s'il décide de ne pas répondre à la question 13, il garde le montant acquis à la question 12, soit 50000 €.

Il y aurait de nombreuses façons de coder ce problème ; en voici une :

La structure Question

Une question est composée du libellé de la question, des 4 libellés pour les réponses et d'une indication de la bonne réponse (un entier de 1 à 4). Par simplicité on en fait une structure mais on pourrait en faire une classe si on voulait par exemple vérifier que la « bonne réponse » possède une valeur correcte.

La structure Gain

Représente un niveau de gain. Elle contient les champs : montant (entier) et palier (un booléen à vrai si cette somme est assurée, faux sinon)

La classe Millionnaire

Cette classe code le moteur du jeu. On y retrouve

- ▷ questionnaire : un tableau de Question
- ▷ gains : un tableau de Gain
- ▷ autres attributs à déterminer (cf. méthodes)

ainsi que les méthodes pour

- ▷ initialiser le jeu à partir d'un questionnaire et du tableau de gains
- ▷ connaitre la question en cours
- ▷ donner la réponse du candidat à la question en cours
- ▷ savoir si le jeu est fini ou pas
- ▷ arrêter le jeu en repartant avec les gains
- ▷ les accesseurs nécessaires pour connaitre l'état du jeu.

Le jeu proprement dit

L'algorithme jeuMillionaireConsole() reçoit le questionnaire et les gains et simule le jeu :

- ▷ Il propose les questions au candidat
- ▷ Il lit ses réponses (chiffre 1 à 4 ou 0 pour arrêter) et fait évoluer le jeu en fonction.
- ▷ lorsque le jeu est terminé, il indique au candidat le montant de ses gains.
- ▷ Attention ! Cet algorithme devrait être le plus petit possible. Imaginez que vous devez également coder une version graphique. Tout code commun doit se trouver dans la classe Millionnaire !

Exercice 7

Chambre avec vue

Un grand hôtel a décidé d'informatiser sa gestion administrative. Il a confié ce travail à la société ESI_INFO dans laquelle vous êtes un informaticien chevronné. On vous a confié la tâche particulière de la gestion des réservations pour ses 100 chambres. Pour ce faire, on vous demande d'écrire une classe Hôtel qui offre notamment une méthode qui permet d'enregistrer une réservation.

Pour représenter l'occupation des chambres un jour donné, nous allons utiliser un tableau de 100 entiers. Un 0 indique que la chambre est libre, une autre valeur (positive) indique le numéro du client qui occupe cette chambre ce jour-là.

Nous utiliserons une Liste de tels tableaux pour représenter l'occupation des chambres sur une longue période ; les éléments se suivant correspondant à des jours successifs.

Nous vous imposons les attributs de la classe, à savoir :

- ▷ occupations : une Liste de tableaux de 100 entiers comme expliqué ci-dessus.
- ▷ premierJour : donne le jour concerné par le premier élément de la liste. Ainsi s'il vaut 10/9/2015 cela signifie que le premier élément de la liste « occupations » renseigne sur l'occupation des chambres ce 10/9/2015 ; que le deuxième élément de la liste concerne le 11/9/2015 et ainsi de suite...

Écrire la méthode suivante

```
method effectuerRéservation(demande ↓ : DemandeRéservation, chambre ↑ : integer) → boolean
```

où la structure de demande de réservation est définie ainsi

```
structure DemandeRéservation
    numéroClient: integer
    débutRéservation: Date
    nbNuitées: integer
end
```

- ▷ Le booléen retourné indique si la réservation a pu se faire ou pas
- ▷ Si elle a pu se faire, le paramètre de sortie chambre indique la chambre qui a été choisie
- ▷ Si plusieurs chambres sont libres, on choisit celle avec le plus petit numéro
- ▷ La demande de réservation peut couvrir une période qui n'est pas encore reprise dans la liste ; il faudra alors l'agrandir

Exercice 8 L'ensemble

La notion d'ensemble fini est une notion qui vous est déjà familière pour l'avoir rencontrée dans plusieurs cours. Nous rappelons certaines de ses propriétés et opérations.

Étant donnés deux ensembles finis **S** et **T** ainsi qu'un élément **x** :

- ▷ $\mathbf{x} \in \mathbf{S}$ signifie que l'élément **x** est un élément de l'ensemble **S**.
- ▷ L'ensemble vide, noté $\emptyset$ est l'ensemble qui n'a pas d'élément ($\mathbf{x} \in \emptyset$ est faux quel que soit **x**).
- ▷ L'ordre des éléments dans un ensemble n'a aucune signification, l'ensemble {1,2} est identique à {2,1}.
- ▷ Un élément **x** ne peut pas être plus d'une fois élément d'un même ensemble (pas de répétition).
- ▷ L'union $\mathbf{S} \cup \mathbf{T}$ est l'ensemble contenant les éléments qui sont dans **S** ou (non exclusif) dans **T**.
- ▷ L'intersection $\mathbf{S} \cap \mathbf{T}$ est l'ensemble des éléments qui sont à la fois dans **S** et dans **T**.
- ▷ La différence $\mathbf{S} \setminus \mathbf{T}$ est l'ensemble des éléments qui sont dans **S** mais pas dans **T**.

Créer la classe Ensemble décrite ci-dessous (où E est le type des éléments de l'ensemble).

```
class Ensemble de E
    public :
        constructor Ensemble de E                          // construit un ensemble vide
        method ajouter(élt : E)                            // ajoute l'élément à l'ensemble
        method enlever(élt : E))                           // enlève un élément de l'ensemble
        method contient(élt : E) → boolean                 // dit si l'élément est présent
        method estVide() → boolean                         // dit si l'ensemble est vide
        method taille() → integer                          // donne la taille de l'ensemble
        method union(autreEnsemble : Ensemble de E) → Ensemble de E
        method intersection(autreEnsemble : Ensemble de E) → Ensemble de E
        method moins(autreEnsemble : Ensemble de E) → Ensemble de E
        method éléments() → List of E                      // conversion en liste
end
```

Quelques remarques :

- ▷ La méthode d'ajout (resp. de suppression) n'a pas d'effet si l'élément est déjà (resp. n'est pas) dans l'ensemble.
- ▷ Les méthodes union(), intersection() et moins() retournent un troisième ensemble, résultat des 2 premiers sans toucher à ces 2 ensembles. On aurait pu envisager des méthodes modifiant l'ensemble sur lequel on les appelle.
- ▷ La méthode éléments() est nécessaire si on veut parcourir les éléments de l'ensemble (par exemple pour les afficher).

Exercice 9 La course à la case 64 en version OO

Reprenez la solution que vous avez écrite pour l'exercice 2 (La course à la case 64 à n joueurs) et voyez comment le transformer pour en faire une version OO.

Exercice 10 Un jeu de poursuite en version OO

Reprenez la solution que vous avez écrite pour l'exercice 5 (Un jeu de poursuite - variante) et voyez comment le transformer pour en faire une version OO.

Exercice 11

Les congés

Les périodes de congés des différents employés d'une firme sont reprises dans un tableau booléen **Congés** bidimensionnel à n lignes et 366 colonnes. Chaque ligne du tableau correspond à un employé et chaque colonne à un jour de l'année. Une case est mise à **vrai** si l'employé correspondant est en congé le jour correspondant. La firme en question est opérationnelle 7 jours sur 7, on n'y fait donc pas de distinction entre jours ouvrables, week-end et jours fériés.

Ce tableau permet de visualiser l'ensemble des congés des travailleurs, et d'accorder ou non une demande de congé, suivant les règles suivantes :

1. une période de congé ne peut excéder 15 jours ;
2. un employé a droit à maximum 40 jours de congé par an ;
3. à tout moment, 50% des employés doivent être présents dans la firme.

Écrire un algorithme qui détermine si cette demande peut être accordée ou non à un employé dont on connait le nom, ainsi que les dates de début et de fin d'une demande de congé (objets de la classe Date). Dans l'affirmative, le tableau **Congés** sera mis à jour.

Pour établir la correspondance entre ce tableau et les noms des employés, vous avez à votre disposition un tableau **Personnel** de chaines. L'emplacement du nom d'un employé dans ce tableau correspond à l'indice ligne du tableau **Congés**.

Il est permis d'utiliser pour résoudre cet exercice la méthode suivante de la classe Date, sans devoir détailler son code :

```
method numéroJour() → integer          // la position du jour dans l'année (entre 1 et 366)
```

Exercice 12

Casino

Pour cet exercice, on vous demande un petit programme qui simule un jeu de roulette très simplifié dans un casino.

Dans ce jeu simplifié, vous pourrez miser une certaine somme et gagner ou perdre de l'argent (telle est la fortune, au casino !). Quand vous n'avez plus d'argent, vous avez perdu.

Notre règle du jeu

Bon, la roulette, c'est très sympathique comme jeu, mais un peu trop compliqué pour un exercice de première année. Alors, on va simplifier les règles et je vous présente tout de suite ce que l'on obtient :

- ▷ Le joueur mise sur un numéro compris entre 0 et 49 (50 numéros en tout). En choisissant son numéro, il y dépose la somme qu'il souhaite miser.
- ▷ La roulette est constituée de 50 cases allant naturellement de 0 à 49. Les numéros pairs sont de couleur noire, les numéros impairs sont de couleur rouge. Le croupier lance la roulette, lâche la bille et quand la roulette s'arrête, relève le numéro de la case dans laquelle la bille s'est arrêtée. Dans notre programme, nous ne reprendrons pas tous ces détails « matériels » mais ces explications sont aussi à l'intention de ceux qui ont eu la chance d'éviter les salles de casino jusqu'ici. Le numéro sur lequel s'est arrêtée la bille est, naturellement, le numéro gagnant.
- ▷ Si le numéro gagnant est celui sur lequel le joueur a misé (probabilité de 1/50, plutôt faible), le croupier lui remet 3 fois la somme misée.
- ▷ Sinon, le croupier regarde si le numéro misé par le joueur est de la même couleur que le numéro gagnant (s'ils sont tous les deux pairs ou tous les deux impairs). Si c'est le cas, le croupier lui remet 50% de la somme misée. Si ce n'est pas le cas, le joueur perd sa mise.

Dans les deux scénarios gagnants vus ci-dessus (le numéro misé et le numéro gagnant sont identiques ou ont la même couleur), le croupier remet au joueur la somme initialement misée avant d'y ajouter ses gains. Cela veut dire que, dans ces deux scénarios, le joueur récupère de l'argent. Il n'y a que dans le troisième cas qu'il perd la somme misée.

Comme vous pouvez le constater, on ne vous fait pas de proposition pour la représentation des données. À vous de jouer !

Exercice 13 Mots croisés

Voici une grille de mots croisés. (on ne s'intéresse pas ici aux définitions). Écrire une classe Grille offrant les méthodes suivantes :

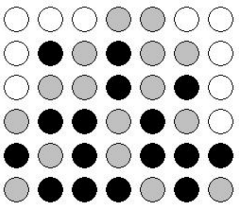
- ▷ placer une lettre à un endroit de la grille (une case non noire bien sûr) ;
- ▷ donner le nombre de cases noires sur la grille ;
- ▷ donner le nombre total de mots (plus d'une lettre) de la grille (donc y compris ceux que le joueur n'a pas encore complétés) ;
- ▷ donner le nombre de mots déjà complétés par le joueur.

Exemple : dans la grille ci-contre, le nombre de cases noires est 14, le nombre total de mots de la grille est 37 (19 horizontaux et 18 verticaux) et le nombre de mots déjà complétés par le joueur est 6.

		A				■			
		L		■					
L	O	G	I	Q	U	E	■		
		O	■				■		
■		R		■		■			
E	S	I	■	O		H		■	
	■	T	A	B	L	E	A	U	■
		H	■	J		B			
		M		E		■			
		E		T					

Exercice 14 Puissance 4

Le jeu de puissance 4 se déroule dans un tableau vertical comportant 6 rangées et 7 colonnes dans lequel deux joueurs introduisent tour à tour des jetons (rouges pour l'un, jaunes pour l'autre). Avec l'aide de la gravité, les jetons tombent toujours le plus bas possible dans les colonnes où on les place. Le jeu s'achève lorsqu'un des joueurs a réussi à aligner 4 de ses jetons horizontalement, verticalement ou en oblique, ou lorsque les deux joueurs ont disposé chacun leur 21 jetons sans réaliser d'alignement (match nul).



N.B. : sur ce dessin noir et blanc, les jetons rouges apparaissent en noir, les jetons jaunes en gris et les cases blanches désignent l'absence de jetons. Cet exemple montre une situation du jeu où le joueur « jaune » est gagnant. En introduisant un jeton dans la 4^{e} colonne, il a réalisé un alignement de 4 jetons en oblique.

On demande d'implémenter une classe Puissance4 qui permette de contrôler l'état des différentes phases du jeu. Déterminez les attributs de cette classe et décrivez-les brièvement de manière à justifier votre choix. Dotez ensuite la classe des méthodes permettant de :

- ▷ savoir si la grille est pleine
- ▷ mettre la grille à jour lorsque le joueur n (1 ou 2) joue dans la colonne j (entre 1 et 7). Cette méthode renverra la valeur booléenne faux si la colonne en question est déjà pleine
- ▷ vérifier si le joueur qui vient de jouer dans la colonne j a gagné la partie

N.B. : pour la structure qui contiendra le contenu du tableau de jetons, on adoptera la convention suivante : 0 pour l'absence de jeton, 1 représentera un jeton du 1^{er} joueur, et 2 un jeton du 2^{e} joueur (on peut donc faire abstraction de la couleur du jeton dans ce problème).

Exercice 15 Mastermind

Revenons sur le jeu Mastermind déjà vu en DEV1. Dans ce jeu, un joueur A doit trouver une combinaison de k pions de couleur, choisie et tenue secrète par un autre joueur B. Cette combinaison peut contenir éventuellement des pions de même couleur. À chaque proposition du joueur A, le joueur B indique le nombre de pions de la proposition qui sont corrects et bien placés et le nombre de pions corrects mais mal placés.

Exemple

Utilisons des lettres pour représenter les couleurs.

Combinaison secrète

R	R	V	B	J

Proposition du joueur

R	V	B	B	V

Il sera indiqué au joueur qu'il a :

- ▷ 2 pions bien placés : le R en 1re position et le second B en 4e position ;
- ▷ 1 pion mal placé : un des deux V (ils ne peuvent compter tous les deux).

Supposons une énumération Couleur (cf. la description d'une énumération en annexe) avec toutes les couleurs possibles de pion.

a) Écrire une classe « Combinaison » pour représenter une combinaison de k pions. Elle possède une méthode pour générer une combinaison aléatoire (que vous ne devez pas écrire) et une méthode pour comparer une combinaison à la combinaison secrète (que vous devez écrire)

b) Écrire ensuite une classe « MasterMind » qui représente le jeu et permet d'y jouer. La taille de la combinaison et le nombre d'essais permis seront des paramètres du constructeur.