Devoir Surveillé 3

16 décembre 2023

INFORMATIQUE MP2I

DURÉE DE L'ÉPREUVE : **4 heures**

L'usage de la calculatrice et de tout dispositif électronique est interdit.

Ce sujet comporte quatre pages numérotées de 1/15 à 15/15

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Tournez la page S.V.P.

Vue d'ensemble du sujet

Ce sujet est composé de 5 parties indépendantes, utilisant les langage de programmation C et OCaml.

Les différentes parties sont indépendantes et peuvent être traités dans n'importe quel ordre. Il n'est pas nécessaire d'avoir répondu aux questions d'une partie pour répondre aux questions d'une autre partie.

- La partie I s'intéresse à la gestion des fichiers dans les systèmes d'exploitations respectant la norme POSIX.
- La partie II s'intéresse à la simulation d'une file d'attente en C, en simulant des variables aléatoires à l'aide de nombres flottants.
- La partie III s'intéresse à des courses de voitures, et à la détermination de stratégies gagnantes en OCaml.
- La partie IV s'intéresse à la représentation compacte de cartes spatiales et à leur lecture en C.
- La partie V s'intéresse à la détermination des gains dans un jeu à gratter en OCaml.

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat. En langage C, il est inutile de rappeler que les entêtes `<assert.h>`, `<stdbool.h>`, etc. doivent être inclus. En OCaml, les noms des arguments des fonctions sont donnés avec des indications de type. Il n'est pas nécessaire de recopier ces indications de type sur votre copie.

Quand l'énoncé demande de coder une fonction, sauf indication explicite de l'énoncé, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

Partie I. Norme POSIX – 10 pts

Ci-dessous, le résultat de la commande linux `ls -al` en ligne de commande sur une installation linux standard :

```
[untel:~/Ag] ls -al
total 36
drwxrwxr-x  3 untel guntel  4096 nov. 7 18:10 .
drwxr-xr-x 75 untel guntel 20480 nov. 7 20:04 ..
lrwxrwxrwx  1 untel guntel     6 nov. 7 18:08 fc.txt -> fi.txt
-rw-rw-r--  1 untel guntel   153 nov. 7 18:10 fi.txt
-rwxrw-r--  1 untel otel      83 nov. 7 18:10 oui.sh
drwxrw-r--  2 untel guntel  4096 nov. 7 18:08 sAg
```

☐ 1 – Quel est l'utilisateur qui utilise le terminal ?

Réponse 1. `untel`

☐ 2 – Quel est le propriétaire des fichiers ?

Réponse 2. `untel`

☐ 3 – Que sont `guntel` et `otel` ?

Réponse 3. Des groupes d'utilisateurs.

☐ 4 – Quel est le nom du répertoire courant ?

Réponse 4. `Ag`, ou `~/Ag`.

☐ 5 – Combien contient-il de fichiers ? de répertoires ?

Réponse 5. Il contient 4 fichiers dont un lien symbolique un répertoire et 2 fichiers standards.

☐ 6 – Pour `oui.sh` et `fc.txt`, expliquer les informations affichées. Des réponses précises sont demandées pour les colonnes 1, 3, 4 ainsi que la dernière.

Réponse 6.

- `oui.sh` : il s'agit d'un fichier standard (`-`), dont l'utilisateur `untel` possède les droits de lecture, d'écriture et d'exécution (`rwx`), les utilisateurs du groupe `otel` possèdent les droits de lecture et d'écriture (`rw-`), et les autres utilisateurs possèdent uniquement le droit de lecture (`r-`). Sa taille en octets est `83`, et sa date de dernière modification est le `nov. 7 18:10`, son nom est `oui.sh`.
- `fc.txt` : il s'agit d'un lien symbolique (`l`), dont l'utilisateur `untel` possède les droits de lecture, d'écriture et d'exécution (`rwx`), les utilisateurs du groupe `guntel` possèdent les droits de lecture, d'écriture et d'exécution (`rwx`), et les autres utilisateurs possèdent les droits de lecture, d'écriture et d'exécution (`rwx`). Sa taille en octets est `6`, et sa date de dernière modification est le `nov. 7 18:08`, son nom est `fc.txt`, et il cible le fichier `fi.txt`.

Nous notons que pour les deux fichiers, la colonne après les permissions indiquent qu'il n'existe qu'un seul lien physique vers ces fichiers.

Partie II. Simulation de file d'attente (C) – 20 pts

Dans toute cette partie, on suppose que l'on dispose d'une fonction `double rand_double()` qui renvoie un nombre flottant aléatoire entre 0 et 1. On rappelle l'existence des fonctions `exp` et `log` qui calculent respectivement l'exponentielle et le logarithme népérien d'un nombre flottant.
Une loi exponentielle modélise la durée de vie d'un phénomène sans mémoire, ou sans vieillissement, ou sans usure : la probabilité que le phénomène dure au moins $s+t$ heures (ou n'importe quelle autre unité de temps) sachant qu'il a déjà duré t heures sera la même que la probabilité de durer s heures à partir de sa mise en fonction initiale. La loi exponentielle de paramètre λ modélise le temps qu'il faut attendre pour voir arriver un phénomène qui intervient avec une fréquence λ.
La méthode de la transformée inverse permet de donner une façon simple de générer un tirage aléatoire suivant une loi exponentielle :

> Si U suit une loi uniforme sur $[0;1]$, alors $E = -\frac{1}{\lambda}\ln(U)$ suit une loi exponentielle de paramètre λ.

☐ 7 – Écrire une fonction `double expo(double lambda)` qui simule un tirage aléatoire selon une loi exponentielle de paramètre λ.

Réponse 7.

C
```c
double expo(double lambda){
  double coins = rand_double();
  return -1./lambda*log(coins);
}
```

Si on suppose qu'un médecin traite en moyenne quatre patients par heure, on modélise le temps pris par une consultation (en heure) par une loi exponentielle de paramètre 4.

☐ 8 – Écrire une fonction `double traite_patients(int n)` qui prend en entrée un nombre entier n qui désigne le nombre de patients et qui calcule le temps pris par le médecin pour traiter les n patients en effectuant une simulation qui modélise le temps pris par chaque consultation (en heure) par une loi exponentielle de paramètre $\lambda = 4$.

Réponse 8.

C
```c
double traite_patients(int n){
  double total = 0.;
  for (int i = 0; i < n; i = i + 1){
    total = total + expo(4.);
  }
  return total;
}
```

En théorie des probabilités et en statistiques, la loi de Poisson est une loi de probabilité discrète qui décrit le comportement du nombre d'événements se produisant dans un intervalle de temps fixé, si ces événements se produisent avec une fréquence moyenne ou espérance connue, et indépendamment du temps écoulé depuis l'événement précédent.
Si le nombre moyen d'occurrences dans un intervalle de temps fixé est γ, alors la probabilité qu'il existe exactement k occurrences ($k \in \mathbb{N}$) est

$$p(k) = \frac{\gamma^k}{k!}e^{-\gamma}$$

Par exemple, si un certain type d'événements se produit en moyenne 4 fois par minute, pour étudier le nombre d'événements se produisant dans un laps de temps de 10 minutes, on choisit comme modèle une loi de Poisson de paramètre $\gamma = 10 \times 4 = 40$.

☐ 9 – Pour quelles valeurs de γ la loi de Poisson de paramètre γ est-elle bien définie ?

Réponse 9. Il faut que $\gamma \geq 0$. En effet, si $\gamma < 0$, $p(1)$ est négatif et ne définit pas une distribution de probabilité.

Voici un algorithme qui permet de simuler une loi de Poisson de paramètre γ en pseudo-code :

- $k \leftarrow 0, p \leftarrow 1$
- tant que $p > e^{-\gamma}$
  - on tire u selon un tirage aléatoire uniforme sur $[0;1]$
  - $p \leftarrow p \times u$
  - $k \leftarrow k + 1$
- on renvoie $k - 1$

☐ 10 – On dit qu'un ensemble ordonné est bien fondé s'il n'existe pas de suite strictement décroissante infinie. Montrer que l'ensemble $[e^{-\gamma};1]$, muni de l'ordre usuel sur les réels n'est pas bien fondé. En déduire que cet algorithme peut ne pas terminer.

Réponse 10. La suite $u : n \mapsto e^{-\gamma} + \frac{1}{n}$ est strictement décroissante et infinie dans $[e^{-\gamma};1]$ à partir d'un certain rang.

☐ 11 – Expliquer pourquoi il est impossible de déterminer un algorithme qui tire un nombre selon une loi de Poisson de paramètre $\gamma > 0$ dont le temps d'exécution est toujours fini.

Réponse 11. En n étapes, un algorithme ne peut séparer que 2^n états possibles. Or, tout entier naturel possède une probabilité non-nulle d'être tirée si $\gamma > 0$, donc pour tout n, n étapes sont insuffisantes.

☐ 12 – Écrire une fonction `int poisson(double gamma)` qui simule un tirage aléatoire selon une loi de Poisson de paramètre γ.

Réponse 12.

C

```c
int poisson(double gamma){
  int k = 0;
  double p = 1;
  while (p > exp(-gamma)){
    double u = rand_double();
    p = p * u;
    k = k + 1;
  }
  return k-1;
}
```

Si on suppose que le médecin accepte trois rendez-vous par heure en moyenne, le nombre de patients qui arrivent en une heure est modélisé par une loi de poisson de paramètre 3.

☐ 13 – Écrire une fonction `int arrivent_patients(double duree)` qui prend en entrée une durée en heures, et qui calcule le nombre de patients qui arrivent sur cette durée en effectuant une simulation qui modélise le nombre de patients qui arrivent pendant une heure par une loi de Poisson de paramètre $\gamma = 3$.

Réponse 13.

C
```c
int arrivent_patients(double duree){
  double param = duree * 3;
  return poisson(param);
}
```

☐ 14 – Écrire une fonction `int nouveaux_patients(int n)` qui calcule le nombre de nouveaux patients qui arrivent pendant la durée que prend le médecin pour traiter n patients.

Réponse 14.

C
```c
int nouveaux_patients(int n){
  double duree = traite_patients(n);
  int nouv_patients = arrivent_patients(duree);
  return nouv_patients;
}
```

☐ 15 – Écrire une fonction `double duree_journee()` qui calcule la durée d'une journée de travail pour le médecin, sachant que les patients commencent à arriver une heure avant que le médecin ne commence ses consultations, et qu'il s'arrête après une consultation dès que plus aucun patient n'attend.

Réponse 15.

C
```c
double duree_journee(){
  int patients = arrivent_patients(1.);
  double journee = 0;
  while (patients > 0){
    double duree = traite_patients(patients);
    journee = journee + duree;
    patients = arrivent_patients(duree);
  }
  return journee;
}
```

Partie III. Courses de voitures (OCaml) – 25 pts

Vous participez à une course de voitures électriques dont l'objectif est le suivant :
- La durée `t` de la course est fixée ;
- l'objectif est de faire aller sa voiture le plus loin possible.

Le contrôle de la voiture est très simple, il n'y a qu'un seul bouton. Plus on appuie sur le bouton, plus la voiture va vite. Cependant, elle ne part que lorsque ce bouton est relâché. Chaque milliseconde où le bouton est appuyé, la vitesse augmente de 1 millimètre par milliseconde (soit 1 mètre par seconde).
Voici un exemple, si la course dure 7 millisecondes, on dispose de 8 choix :

- Maintenir le bouton enfoncé 0 millisecondes. Dans ce cas, la voiture reste immobile et elle parcourt 0 millimètres.
- Maintenir le bouton enfoncé 1 milliseconde. Dans ce cas, la voiture se déplace à 1m.s^{-1} pendant 6 millisecondes et parcourt 6 millimètres.
- Maintenir le bouton enfoncé 2 millisecondes. Dans ce cas, la voiture se déplace à 2m.s^{-1} pendant 5 millisecondes et parcourt 10 millimètres.
- etc.

☐ 16 – Compléter l'exemple. (Il n'est pas nécessaire de recopier les phrases en entier, on veut juste la correspondance entre le choix effectué et la distance parcourue)

Réponse 16.

- Maintenir le bouton enfoncé 3 millisecondes : 12 millimètres parcourus.
- Maintenir le bouton enfoncé 4 millisecondes : 12 millimètres parcourus.
- Maintenir le bouton enfoncé 5 millisecondes : 10 millimètres parcourus.
- Maintenir le bouton enfoncé 6 millisecondes : 6 millimètres parcourus.
- Maintenir le bouton enfoncé 7 millisecondes : 0 millimètres parcourus.

☐ 17 – Écrire une fonction `score (n : int) (t : int) : int` qui prend en entrée la durée n durant laquelle le bouton est appuyé et t la durée totale de la course et qui renvoie la distance parcourue par la voiture.

Réponse 17.

```
let score n t = n * (t - n)
```

☐ 18 – Écrire une fonction `best (t : int) : int` qui prend en entrée la durée totale t de la course et qui renvoie la plus grande distance qu'il est possible d'atteindre.

Réponse 18.

```
let best t = score (t/2) t
```

On s'intéresse maintenant au nombre de manières possible de battre le record `d` de distance pour une course donnée. Par exemple, pour la course présentée précedemment, si le record `d` est de 9 millimètres, il est possible de le dépasser de 4 manières différentes, la plus petite solution étant d'appuyer 2 millisecondes sur le bouton. On propose la solution suivante, en OCaml :

```
let possibilities t d =
  let cpt = ref 0 in
  for n = 0 to t do
    if score n t > d then
      incr cpt;
  done; !cpt
```

On rappelle que si `x` est une valeur entière, `ref x` désigne une nouvelle variable entière (type `int ref`) dont la valeur est initialisée à `x`. Si `var` est de type `int ref`, on peut accéder à sa valeur avec l'expression `!var`, et on peut modifier sa valeur avec l'opérateur `:=`, ou avec les fonctions d'incrémentation `incr` (équivalent à `var := !var + 1`) et de décrémentation `decr` (équivalent à `var := !var - 1`).

☐ 19 – Donner un ou plusieurs invariants de la boucle `for` qui permet de justifier la correction partielle de cette fonction.

Réponse 19. La variable désignée par `cpt` contient le nombre de manières possibles de battre le record en appuyant moins de `n` millisecondes sur le bouton.

☐ 20 – Donner la complexité de la fonction `possibilities` en fonction de `t`, en utilisant une notation asymptotique O, et en justifiant brièvement. On pourra supposer que la fonction `score` s'exécute en temps $O(1)$ dans le pire des cas.

Réponse 20. Chaque itération de la boucle s'effectue en temps constant $O(1)$. La boucle effectue `t+1` itérations. La complexité globale est donc $O(t)$.

Les courses étant trop longues et la complexité de cette fonction étant trop élevée, on cherche à trouver le résultat plus rapidement.

☐ 21 – On suppose que l'on a déjà calculé la durée minimale $n_{\min}$ durant laquelle il faut appuyer sur le bouton pour battre le record de distance d pour une course de durée totale t. Donner la durée maximale $n_{\max}$ durant laquelle il faut appuyer sur le bouton pour battre ce même record, et en déduire le nombre de manières possibles de battre le record.

Réponse 21. $n_{\max} = t - n_{\min}$, donc il y a $t - 2n_{\min} + 1$ manières possibles de battre le record.

Voici une fonction qui calcule le nombre de manières possibles de battre le record, en testant une à une les possibilités et en s'arrêtant dès qu'elle a trouvé une solution qui bat le record.

```
let possibilities_rec t d =
  (* À compléter : assertion *)
  let rec aux i =
    if (* À compléter : 1 *) then i
    else (* À compléter : 2 *)
  in
  let nmin = aux 0 in
  (* À compléter : 3 *)
```

☐ 22 – Compléter le code de la fonction `possibilities_rec` en remplaçant les commentaires 1, 2, et 3.

Réponse 22.

```
let possibilities_rec t d =
  let rec aux i =
    if score i t > d then i
    else aux (i+1)
  in
  let nmin = aux 0 in
  t-2*nmin+1
```

☐ 23 – Si le record d est inatteignable, cette fonction boucle infiniment (si on ignore les dépassement de capacité). Remplacer le commentaire restant par une assertion permettant de vérifier que le record est battable avant de rechercher le nombre de manières de le faire.

Réponse 23. `assert (best t > d);`

☐ 24 – Justifier que cette fonction est de complexité $O(t)$ dans le pire des cas.

Réponse 24. Si la seule possibilité pour dépasser le record d est d'appuyer sur le bouton $t/2$ millisecondes, alors le nombre d'appels récursifs de la fonction `aux` est $t/2 = O(t)$.

☐ 25 – En supposant que le record est battable, donner un encadrement de la valeur $n_{\min}$.

Réponse 25. $0 \leq n_{\min} \leq \frac{t}{2}$.

☐ 26 – Écrire une fonction `nmin (t : int) (d : int) : int` qui calcule la valeur $n_{\min}$ en temps $O(\log t)$.

Réponse 26. On procède par dichotomie :

```
let nmin t d =
  let rec aux min max =
    if max - min = 1 then max
    else
      let mid = (min + max) / 2 in
      if score mid t > d then aux min mid
      else aux mid max
  in aux 0 (t/2)
```

Partie IV. Observatoire (C) – 20 pts

On cherche à utiliser les données d'un observatoire pour calculer les distances entre des galaxies. Voici le format que l'observatoire utilise pour représenter ses données sur un exemple, où # représente une galaxie et . du vide :

```
....#........
.........#...
#............
............
............
........#....
.#...........
............#
............
............
.........#...
#....#.......
```

En C, ceci est représenté par une matrices de caractères `char exemple[12][13]`, où la case `exemple[i][j]` contient `'#'` si une galaxie est présente et `'.'` sinon. La première coordonnée est donc l'ordonnée (qui croît en descendant), et la seconde l'abscisse (qui croît vers la droite).
On utilisera la distance de Manhattan. Par exemple, sur la figure suivante, on peut voir que la distance entre la galaxie 5 et la galaxie 9 est 9.

```
....1........
.........2...
3............
............
............
........4....
.5...........
.##.........6
..##.........
...##........
....##...7...
8....9.......
```

☐ 27 – Écrire une fonction `int abs(int x)` qui renvoie la valeur absolue d'un entier.

Réponse 27.

C
```c
int abs(int x){
  if (x < 0) { return -x; }
  return x;
}
```

☐ 28 – Écrire une fonction `int distance(int i1, int j1, int i2, int j2)` qui calcule la distance entre les points de coordonnées

Réponse 28.

C
```c
int distance(int i1, int j1, int i2, int j2){
  int di = abs(i1-i2);
  int dj = abs(j1-j2);
  return di + dj;
}
```

Pour compresser les données, l'univers étant lacunaire, l'observatoire possède 2 échelles différentes. Chaque ligne ou colonne qui ne contient aucune galaxie représente en réalité un million (1 000 000) de lignes ou de colonnes. Pour le reste de l'exercice, on supposera que les cartes de l'observatoire ont pour dimension 400×600, pour pouvoir les représenter en C par des tableaux de taille fixe.

☐ 29 – Écrire une fonction `bool est_vide(char data[400][600], int i)` qui renvoie `true` si la ligne `i` ne contient aucune galaxie et `false` sinon.

Réponse 29.

C
```c
void est_vide(char data[400][600], int i){
  for (int j = 0; j < 600; j = j + 1){
    if (data[i][j] == '#')
      { return false; }
  }
  return true;
}
```

☐ 30 – Écrire une fonction `void real_i(int result[400], char data[400][600])` qui prend en entrée un tableau d'entiers de dimension 400 qui n'est pas forcément initialisé, et une carte de l'observatoire de dimension 400×600, et qui calcule pour chaque indice i l'ordonnée de la dernière ligne représentée par la ligne i de la carte, et en remplit le tableau `result`. Si on lance la fonction sur l'exemple, on aurait comme résultat dans le tableau `result` :
`{0, 1, 2, 1000002, 2000002, 2000003, 2000004, 2000005, 3000005, 4000005, 4000006, 4000007}`

Réponse 30.

C
```c
void real_i(int result[400], char data[400][600]){
  if (est_vide(data, 0))
    { result[0] = 1; }
  else
    { result[0] = 1000000; }
  for (int i = 1; i < 400; i = i + 1){
    if (est_vide(data, i))
      { result[i] = result[i-1] + 1; }
    else
      { result[i] = result[i-1] + 1000000; }
  }
}
```

On suppose qu'on a de même écrit une fonction similaire `real_j` pour travailler sur les colonnes.

☐ 31 – Compléter le code de la fonction suivante, qui calcule la somme des distances entre chaque paire de galaxies.

```
int sum_distances(data[400][600]){
  int ri[400], rj[600];
  real_i(ri, data); real_j(rj, data);
  /* À compléter : 1 */
  for (int i = 0; i < 400; i = i + 1){
    for (int j = 0; j < 600; j = j + 1){
      if (/* À compléter 2 */){
        for (int ii = 0; ii < 400; ii = ii + 1){
          for (int jj = 0; jj < 600; jj = jj + 1){
            if (/* À compléter 3 */){
              /* À compléter 4 */
            }
          }
        }
      }
    }
  }
  /* À compléter 5 */
}
```

Réponse 31.

C

```
int sum_distances(data[400][600]){
  int ri[400], rj[600];
  real_i(ri, data); real_j(rj, data);
  int cpt = 0;
  for (int i = 0; i < 400; i = i + 1){
    for (int j = 0; j < 600; j = j + 1){
      if (data[i][j] == '#'){
        for (int ii = 0; ii < 400; ii = ii + 1){
          for (int jj = 0; jj < 600; jj = jj + 1){
            if (data[ii][jj]){
              cpt = cpt + dist(ri[i],rj[j],ri[ii],rj[jj]);
            }
          }
        }
      }
    }
  }
  return cpt / 2;
}
```

☐ 32 – Donner la complexité de cette fonction, en fonction de la hauteur h de la carte et la largeur w de la carte. On pourra supposer que la fonction `real_i` a pour complexité $O(h)$ et `real_j` a pour complexité $O(w)$.

Réponse 32. $O(w^2h^2)$. Les 4 boucles imbriquées dominent le temps d'exécution, et chaque itération de la boucle la plus itérieure se fait en temps constant.

☐ 33 – En sachant que la plupart des cases ne sont pas des galaxies, on peut résoudre ce problème plus efficacement qu'en parcourant tous les couples de cases.
Proposer une méthode plus efficace lorsque le nombre de galaxie n et petit par rapport au nombre de cases $h \times w$. Estimer sa complexité en fonction de n, h, et w.
On ne demande pas d'écrire de code en C.

Réponse 33. On pourrait suivre cette méthode :

1. parcourir une fois toutes les données et retenir dans un ou plusieurs tableaux les coordonnées des galaxies (complexité $O(wh)$) ;
2. parcourir les couples de galaxies grâce à cette nouvelle information (complexité $O(n^2)$).

Pour une complexité totale de $O(wh + n^2)$. Notons que dans le pire des cas, $n = wh$ et on obtient la même complexité asymptotique, mais que si n est plus petit (e.g. $\sqrt{wh}$), on peut y gagner beaucoup ($O(n^2)$ au lieu de $O(n^4)$).

Partie V. Jeux à gratter (OCaml) – 20 pts

Sur une carte de jeux à gratter, on trouve deux listes de nombres :
- une liste de numéros gagnants ;
- une liste de numéros obtenus.

Pour déterminer les gains d'une carte, il faut compter le nombre de numéros obtenus qui sont dans la liste des numéros gagnants :
- si un des numéros obtenus se trouve dans la liste des numéros gagnants, la grille rapporte un point ;
- pour chacun des numéros obtenus qui se trouve dans la liste des numéros gagnants en plus du premier, les points sont doublés.

Par exemple, si les numéros gagnants sont `41 48 83 86 17`, et que les numéros obtenus sont `83 86 6 31 17 9 48 53`, la carte rapporte 2 points pour les numéros `83` et `48`.

☐ 34 – Écrire une fonction `score (n : int) : int` qui calcule en fonction du nombre de numéros obtenus qui sont dans la liste des numéros gagnants le nombre de points gagnés par la carte.

Réponse 34.

```
let rec score n =
  if n < 2 then n
  else 2 * score (n-1)
```

Les cartes sont représentés en OCaml par 2 tableaux d'entiers, chacun de type `int array`. L'exemple ci-dessus sera représenté en OCaml par :

```
let example_winning = [|41;48;83;86;17|]
let example_gotten = [|83;86;6;31;17;9;48;53|]
```

On rappelle les syntaxes suivantes relatives aux tableaux en OCaml :

- `tab.(i)` accède à la case i du tableau `tab` ;
- `tab.(i) <- x` affecte la valeur x à la case i du tableau `tab` ;
- `Array.length : 'a array -> int` donne la longueur d'un tableau ;
- `Array.make : int -> 'a -> 'a array` à partir d'un entier n et d'une valeur x crée un nouveau tableau de taille n dont toutes les cases sont initialisées à x ;
- `Array.init : int -> (int -> 'a) -> 'a array` à partir d'un entier n et d'une fonction f crée un nouveau tableau de taille n dont la case i est initialisé avec la valeur $f(i)$.

☐ 35 – Écrire une fonction `mem (x : 'a) (tab : 'a array) : bool` qui renvoie `true` si x est dans le tableau `tab` et `false` sinon. La complexité de cette fonction doit être $O(n)$ dans le pire des case où n est la taille du tableau.

Réponse 35.

```
exception Found
let mem x tab =
  try
    for i = 0 to Array.length tab - 1 do
      if tab.(i) = x then raise Found
    done;
    false
  with Found -> true
```

☐ 36 – Écrire une fonction `fold_left (f : 'a -> 'b -> 'a) (acc : 'a) (tab :'b array) -> 'a` qui étant donné une fonction `f`, une valeur initiale d'accumulateur `acc` et un tableau `t = [|a1;a2;...:an|]` calcule `f (... (f (f acc a1) a2) ...) an`.

Réponse 36.

```
let fold_left f acc tab =
  let rec aux i acc =
    if i = Array.length tab then acc
    else aux (i+1) (f acc)
  in aux 0 acc
```

☐ 37 – En déduire une fonction `card_score (winnings : int array) (gotten : int array) : int` qui calcule le score d'une carte en parcourant le tableau `gotten` des numéros obtenus et en comptant combien sont dans le tableau `winnings` des numéros gagnants.

On pourra utiliser le mot clé `fun` pour définir une fonction anonyme comme dans l'exemple `(fun x -> 2 * x)` qui désigne la fonction qui double son entrée, ou encore `(fun x y -> x + y)` qui désigne la fonction `(+)` effectuant l'addition de deux entiers.

Réponse 37.

```
let card_score winnings gotten =
  let f acc x = if mem x winnings then acc + x else acc in
  let n = fold_left f 0 gotten in
  score n
```

☐ 38 – Donner la complexité de cette approche en fonction de la taille w du tableau des nombres gagnants, g du tableau des nombres obtenus, et n le nombre de nombres obtenus qui sont dans les nombres gagnants.

Réponse 38. $O(wg + n) = O(wg)$ car $n \leq w$ et $n \leq g$: on teste chaque couple de nombre gagnant/nombre obtenu.

On suppose à présent que l'on dispose d'un tableau de booléens `tmp : bool array` dont la taille est plus grande que tous les nombres pouvant apparaître sur les cartes, et dont toutes les cases sont initialisées à `false`.

☐ 39 – Écrire une fonction `put_in (gotten : int array) (tmp : bool array) : unit` qui met à `true` les valeurs de `tmp.(i)` pour chaque nombre `i` présent dans le tableau `gotten`.

Réponse 39.

```
let put_in gotten tmp =
  fold_left (fun () i -> tmp.(i) <- true) () gotten
```

☐ 40 – Écrire une fonction `count_out (winnings : int array) (tmp : bool array) : int` qui renvoie le nombre de cases `tmp.(i)` du tableau `aux` valant `true` telles que `i` est dans le tableau `winnings`.

Réponse 40.

```
let count_out winnings tmp =
  fold_left (fun acc i -> if tmp.(i) then acc + 1 else acc) 0 winnings
```

☐ 41 – En déduire une fonction

`card_score_opt (winnings : int array) (gotten : int array) (aux : bool array) : int`.

Donner la complexité de cette nouvelle approche.

Réponse 41.

```
let card_score_opt winnings gotten aux =
  put_in gotten aux;
  let n = count_out winnings aux in
  score n
```

$O(g + w)$.

En pratique, cette seconde méthode serait implémentée à l'aide de tableaux associatifs, pouvant être implémentés par des tables de hachages, ou encore des arbres binaires de recherche, pour éviter d'avoir à initialiser le tableau `aux`, opération qui dominerait potentiellement le temps d'exécution de la fonction.

FIN DE L'ÉPREUVE