Devoir Surveillé 3

14 décembre 2024

INFORMATIQUE MP2I

DURÉE DE L'ÉPREUVE : **4 heures**

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

*Ce sujet comporte huit pages numérotées de 1/**??** à **??**/**??***

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 l'analyse d'une solution d'un problème classique appelé 'Dutch Flag' en C, via des invariants.
- 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. Drapeau de la Pologne (C – 5 pts)

On considère l'algorithme suivant :

Polish national flag — C

```c
void polish(int taille, int tab[]) {
  // Précondition : tab[] est un tableaux d'entiers de taille n
  // contenant uniquement des 0 et des 1
  int i = 0, j = taille-1;
  while (j > i) {
  /****************************************
   *                  i     j             *
   *       +-+-+---+-+-+---+-+-+-+---+-+  *
   * tab : |0|0|...|0|?|...|?|1|1|...|1|  *
   *       +-+-+---+-+-+---+-+-+-+---+-+  *
   ****************************************/
   if (0 == tab[i]) {
    i = i + 1;
   } else {
    tab[i] = tab[j];
    tab[j] = 1;
    j = j - 1;
   }
  }
}
```

☐ 1 – Expliciter l'invariant de boucle expliqué par le commentaire.

Réponse 1. $\forall k < i$, `tab[k]` $= 0, \forall k > j$, `tab[k]` $= 1$

☐ 2 – Soit n le nombre de 0 et m le nombres de 1 dans le tableau au début de la fonction. Montrer l'invariant de boucle suivant :

Le tableau contient n fois 0 et m fois 1.

Réponse 2.

- L'invariant est vérifié au début de la première itération par définition de n et m
- En supposant que l'invariant est vérifié au début d'une itération, on a deux cas à vérifier :
  - Si `(0 == tab[i])`, le tableau est inchangé
  - Sinon, `tab[i]` vaut 1, et le nombre de 0 et de 1 reste inchangé car on a juste échangé les valeurs de `tab[i]` et de `tab[j]`.

☐ 3 – Que fait cette fonction ?

Réponse 3. Cette fonction trie le tableau en mettant les 0 avant les 1.

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 qui modélise un tirage aléatoire uniforme entre 0 et 1 exclus. 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.

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

Réponse 4.

```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 (l'espérance d'une loi exponentielle de paramètre λ est $\frac{1}{\lambda}$).

☐ 5 – 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 5.

```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 $\gamma = 10 \times 4 = 40$.

☐ 6 – Pour quelles valeurs de γ la loi de Poisson de paramètre γ est-elle bien définie ?
On pourra supposer que $\forall \gamma \in \mathbb{R}, \sum_{k=0}^{+\infty} \frac{\gamma^k}{k!} = e^{\gamma}$.

Réponse 6. 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 $k - 1$

☐ 7 – 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 si on suppose que les variables sont des nombres réels.

Réponse 7. 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.
Il est possible que la variable p prenne les valeurs successives de cette suite et donc ne soit jamais inférieur ou égal à $e^{-\gamma}$.

☐ 8 – 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 dans le pire des cas est borné par une constante M.

Réponse 8. En n étapes, un algorithme ne peut séparer que 2^n états possibles. En particulier, aucun algorithme ne peut écrire le résultat 2^n en n étapes si $n > M$. Or, tout entier naturel possède une probabilité non-nulle d'être tirée si $\gamma > 0$, en particulier M.

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

Réponse 9.

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, et de manière équivalente, le nombre de patients qui arrivent en 20 minutes est modélisé par une loi de poisson de paramètre 1.

☐ 10 – 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 10.

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

☐ 11 – 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 11.

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

☐ 12 – 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 12.

```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.

☐ 13 – 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 13.

— 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.

☐ 14 – 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 14.

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

☐ 15 – 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. Justifier sa correction.

Réponse 15. La fonction $n \mapsto n(t-n)$ est un polynôme du second degré dont le maximum est atteint en $t/2$. Si t est impair, les valeurs en $(t-1)/2$ et $(t+1)/2$ sont égales.

```
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`).

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

Réponse 16. La variable désignée par `cpt` contient le nombre de manières possibles de battre le record en appuyant moins de `n` millisecondes sur le bouton.

☐ 17 – 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 17. 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.

☐ 18 – 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 18. Par symétrie de la parabole, $n_{\max} = t - n_{\min}$, 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 *)
```

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

Réponse 19.

```
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
```

☐ 20 – 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 20. `assert (best t > d);`

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

Réponse 21. 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 $t/2 = O(t)$.

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

Réponse 22. $0 \leq n_{\min} \leq \frac{t}{2}$.

☐ 23 – Écrire une fonction `nmin (t : int) (d : int) : int` qui calcule la valeur $n_{\min}$ en temps $O(\log t)$. *Il existe une solution qui a pour complexité $O(1)$ dans le pire des cas.*

Réponse 23. En $O(\log t)$, 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)
```

Ce n'est pas la solution la plus efficace. On recherche en effet les racines d'un polynôme du second degré $n \mapsto n(t-n) - d$. Calculer ses racines à l'aide d'un discriminant et les renvoyer se fait en $O(1)$.

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.......
```

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

Réponse 24.

C

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

☐ 25 – Écrire une fonction `int distance(int i1, int j1, int i2, int j2)` qui calcule la distance entre les points de coordonnées (i_1, j_1) et (i_2, j_2).

Réponse 25.

```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.

☐ 26 – 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 26.

```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;
}
```

☐ 27 – 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 27.

```c
void real_i(int result[400], char data[400][600]){
  if (est_vide(data, 0))
    { result[0] = 0; }
  else
    { result[0] = 999999; }
  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.

☐ 28 – 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 28.

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;
}
```

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

☐ 30 – 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 30. 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 8 points pour les numéros `83`, `86`, `48`, et `17`.

☐ 31 – 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 31.

```
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 $f(i)$.

☐ 32 – 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 32.

```
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
```

Ou encore

```
let mem x tab =
  let n = Array.length tab in
  let rec search i =
    i < n && (tab.(x) = i || search (i+1))
  in
  search 0
```

☐ 33 – 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 33.

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

Ou encore

```
let fold_left f acc tab =
  let var_acc = ref acc in
  for i = 0 to Array.length tab - 1 do
    var_acc := f !var_acc tab.(i)
  done;
  !var_acc
```

☐ 34 – 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 34.

```
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
```

☐ 35 – 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 35. $O(wg + n) = O(wg)$ 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`.

☐ 36 – 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 36.

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

☐ 37 – 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 37.

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

☐ 38 – En déduire une fonction

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

Donner la complexité de cette nouvelle approche.

Réponse 38.

```
let card_score_opt winnings gotten tmp =
  put_in gotten tmp;
  let n = count_out winnings tmp 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 `tmp`, opération qui dominerait potentiellement le temps d'exécution de la fonction.

Fin de l'épreuve