Informatique
MP2I 2022/2023
DS2 - Correction
CIV

Durée totale 4h.
Tout document interdit. Tout dispositif électronique interdit (calculatrice, cafetière, Francky...)
On vous demande d'écrire lisiblement et de bien présenter. Les programmes en particulier doivent être bien indentés et lisibles.
Bon courage et *à l'abordage!*

À la recherche du M-PIece



Remarque 1. *Les preuves de correction ou de terminaison présentée ici sont un objectif de rédaction à atteindre. Des réponses partielles ou moins détaillées sont acceptées.*

1 Passer en revue l'équipage

Avant de partir, c'est le moment de passer en revue l'équipage et de s'assurer que tout le monde est qualifié.
Plusieurs membres de l'équipage vous montrent ainsi des programmes qu'ils ont réalisés et vous allez devoir comprendre ce qu'ils font et montrer qu'ils sont corrects.
On considère le programme C suivant :

```
bool palindrome(char str[]){
    int i = 0,
    j = 0;
    while (str[j] != '\0') {
        j = j + 1;
    }
    j = j - 1;
    while (j >= i) {
        if (str[i] != str[j]) {
            return false;
        }
        i = i+1;
        j = j-1;
    }
    return true;
}
```

On suppose que `str` sera toujours une chaîne de caractère valide en C.

Q.1

1. Dérouler l'exécution de ce programme sur les chaînes `"kayak"` et `"canoe"`.
2. Montrer que toutes les boucles terminent.
3. Montrer que ce programme est correct, c'est-à-dire qu'il renvoie un booléen indiquant si une chaîne est un

palindrome.

Indication : On rappelle qu'un *palindrome* est une chaine de caractère pouvant se lire de la même manière de gauche à droite et de droite à gauche.

R.1

Attention erreur d'énoncé ici, il fallait bien penser à incrémenter i et décrémenter j.

1. — Pour `"kayak"`, première boucle :

j	$str[j]$
0	$'k'$
1	$'a'$
2	$'y'$
3	$'a'$
4	$'k'$
5	0

ensuite on passe j à 4 et on entre dans la seconde boucle.

i	j	$str[j]$	$str[i]$
0	4	$'k'$	$'k'$
1	3	$'a'$	$'a'$
2	2	$'y'$	$'y'$
3	1		

Comme on n'est pas sortie avant, on renvoie `true`.

— Pour `"canoe"`, première boucle :

j	$str[j]$
0	$'c'$
1	$'a'$
2	$'n'$
3	$'o'$
4	$'e'$
5	0

ensuite on passe j à 4 et on entre dans la seconde boucle.

i	j	$str[j]$	$str[i]$
0	4	$'c'$	$'e'$

Ici, on sort tout de suite en renvoyant `false`.

2. Comme la chaine est valide, on sait que la caractère `'\0'` est présent à l'indice n. On a alors la quantité $n - j$ qui est un variant de la première boucle car :
   — entière
   — positive tout au long de la boucle vu que la condition implique $j < n$ quand on boucle
   — diminue strictement à chaque itération car j augmente.

   Ainsi la première boucle termine.

   La seconde termine car la quantité $j - i$ est un variant :
   — elle est entière
   — positive tout au long de la boucle car la condition $i \leq j$ implique $j - i \geq 0$.
   — diminue strictement à chaque itération car $j - 1 - (i + 1) = j - i - 2 < j - i$.
3. La première boucle termine et en sortie on a $j = n$ où n est la première occurrence de `'\0'` dans la chaine, c'est-à-dire sa longueur.

   On va montrer que la propriété

$$I(i, j) = n - i - 1 = j \wedge \forall k \in [\![0, i - 1]\!], s[k] = s[n - k - 1]$$

   est un invariant.
   — Initialisation : au départ, $i = 0$ donc la plage $[\![0, i - 1]\!] = \emptyset$ et l'invariant $I(0, n - 1)$ est vérifié.
   — Hérédité : si $I(i, j)$ est vérifié en début d'itération, comme on est rentré dans la boucle, on a $s[i] = s[j] = s[n-1-i]$ donc la propriété d'égalité est vérifiée pour $k = i$, on a donc $\forall k \in [\![0, i]\!]$. De plus, $n-(i+1)-1 = n - i - 1 - 1 = j - 1$ donc $I(i + 1, j - 1)$ est vérifié.

   L'invariant est validé et en sortie de boucle, on a $n - i - 1 = j < i \iff i > \frac{n-1}{2} \geq \lfloor \frac{n-1}{2} \rfloor$ donc $i \geq \lfloor \frac{n}{2} \rfloor$ et l'égalité assure ainsi que la première moitié se retrouve à l'envers dans la seconde moitié : c'est un palindrome.

On considère le programme OCaml suivant :

```
let div a b =
    let q = ref 0 in
    let r = ref a in
    while !r >= b do
        r := !r - b;
        q := !q + 1
    done;
    (!q, !r)
```

On suppose ici que a, b sont des entiers naturels et que $b > 0$.

Q.2

1. Dérouler l'exécution de ce programme pour $a = 42$ et $b = 4$.
2. Montrer que ce programme termine.
3. Montrer que le couple (q, r) renvoyé vérifie $a = bq + r$ et $0 \leq r < b$.

R.2

1.

q	r
0	42
1	38
2	34
3	30
4	26
5	22
6	18
7	14
8	10
9	6
10	2

2. La quantité r est un variant :
   - elle est entière (uniquement des soustractions)
   - est est positive tout au long de la boucle car si on rentre dans une itération c'est que $r \geq b$, donc en fin d'itération $r - b \geq 0$ et c'est la nouvelle valeur de r
   - comme $b > 0$, elle diminue strictement à chaque itération.

   Ainsi, comme elle admet un variant, la boucle termine.
3. On a l'invariant $I(q, r) \cong a = bq + r$:
   - Initialisation : $a = b \times 0 + a$ donc $I(0, a)$ est vérifié
   - Itération : si $I(q, r)$ est vérifié en début d'itération, alors $a = bq + r = b(q + 1) + r - b$ donc $I(q + 1, r - b)$ est vérifié en fin d'itération ce qui correspond aux nouvelles valeurs de (q, r).

   Ainsi, en sortie de boucle, on a $a = bq + r$ et comme on est sorti $r < b$. De plus, on a vu à la question précédente que $r \geq 0$.

2 Trouver la bonne vitesse - C

Afin de planifier la route, votre navigatrice, Cami, étudie les données qu'elle a *empruntées* à d'autres équipages. Elle a ainsi obtenu une suite de n nombres entiers indiquant la vitesse du bateau à intervalle régulier.
Par exemple, on aurait pu obtenir cette suite exemple de 10 nombres :

$$199 \quad 200 \quad 208 \quad 210 \quad 200 \quad 207 \quad 240 \quad 269 \quad 260 \quad 263$$

Dans toutes les questions, on suppose que cette suite est donnée sous la forme d'un tableau d'entiers du type `int v[]` et du nombre `int n` de vitesses enregistrées.
En premier lieu, on va s'intéresser au nombre de moments où le bateau a accéléré, indiqué ici avec une flèche $\nearrow$:

$$199 \nearrow 200 \nearrow 208 \nearrow 210 \searrow 200 \nearrow 207 \nearrow 240 \nearrow 269 \searrow 260 \nearrow 263$$

On voit sur cet exemple que le bateau a accéléré 7 fois.
Attention : on considère qu'on accélère quand la vitesse augmente **strictement**.

Q.3

Écrire une fonction de prototype

```
int accelerations(int v[], int n);
```

qui prend un tableau de vitesse en entrée et renvoie le nombre de fois où le bateau a accéléré.

R.3

```
int accelerations(int v[], int n) {
    int accel = 0;
    for(int i = 0; i < n-1; i++) {
        if (v[i] < v[i+1]) {
            accel = accel + 1;
        }
    }
    return accel;
}
```

Q.4

Prouvez la correction de votre fonction.
Indication il sera sûrement nécessaire de s'assurer qu'à tout moment, vous avez compté le nombre de fois où le bateau a accéléré **dans la partie étudiée** du tableau.

R.4

Conformément à l'indication, on remarque que `accel` contient toujours le nombre d'accélérations du préfixe étudié du tableau.
On considère ainsi le candidat invariant $I(\texttt{accel}, i) \cong$ le nombre d'accélérations dans $v[0], \ldots, v[i]$ est `accel`. On va le valider :

- Initialisation : au départ, $i = 0$ donc le préfixe contient un seul élément, il ne peut être question d'accélération, et comme initialement $\texttt{accel} = 0$, on a bien $I(\texttt{accel}, i)$.
- Itération : si $I(\texttt{accel}, i)$ est vérifié en début d'itération, on a deux cas :
  - $v[i] >= v[i+1]$, alors il n'y pas de nouvelle accéleration et comme on ne change pas `accel`, on a bien $I(\texttt{accel}, i+1)$.
  - $v[i] < v[i+1]$, on a alors une accéleration de plus et ainsi on a bien $I(1 + \texttt{accel}, i+1)$.

  Dans tous les cas, I est vérifié en fin d'itération.

Ainsi, en sortie de boucle $I(\texttt{accel}, n-1)$ est vérifié donc `accel` contient les accélerations du tableau complet.

Q.5

Écrire une suite d'instructions qui pourraient être placées dans la fonction `main` d'un programme pour :

- définir la tableau de l'exemple
- calculer le nombre d'accélérations `N` avec la fonction précédente
- afficher une ligne indiquant `Le bateau a acceleré N fois` où `N` est affiché sous forme décimale.

R.5

```
int v[] = { 199, 200, 208, 210, 200, 207, 240, 269, 260, 263 };
int N = accelerations(v, 10);
printf("Le bateau a acceleré %d fois.\n", N);
```

Dans l'exemple précédent, on peut voir que la bateau accélère au plus trois fois d'affilée avant de ralentir, cela est réalisé deux fois : $199 \nearrow 200 \nearrow 208 \nearrow 210$ et $200 \nearrow 207 \nearrow 240 \nearrow 269$. Une série d'accélération sera appelée ici une **poussée**.

Q.6

Écrire une fonction de prototype

```
int poussee(int v[], int n, int i);
```

où v et n décrivent un tableau vitesse comme précédemment, $0 \leq i < n$ et qui renvoie le nombre de fois où le bateau a acceleré à partir de la vitesse $v[i]$.
Dans l'exemple précédent, pour $i = 0$ on renverra 3, pour $i = 2$ on renverra 1et pour $i = 3$ on renverra 0.

R.6

```
int poussee(int v[], int n, int i) {
    int j = i;
    while(j < n-1 && v[j] < v[j+1]) {
        j += 1;
    }
    return j-i;
}
```

Q.7

Prouver la correction de votre fonction, c'est-à-dire que si elle renvoie N pour i alors la suite de valeurs $v[i], v[i+1], ..., v[i+N]$ est croissante et, soit $i+N+1=n$ soit $v[i+N] \geq v[i+N+1]$.
Attention il sera sûrement nécessaire ici de s'assurer de la terminaison.

R.7

On remarque rapidement que $n-j$ est un variant, donc la boucle termine.
On va valider l'invariant $I(j) \cong \forall k \in [\![i, j-1]\!], v[k] < v[k+1]$.

- Initialisation : au départ $j=i$, donc $[\![i, j-1]\!] = \emptyset$ et $I(j)$ est trivialement vérifié.
- Itération : si on a $I(j)$ en début d'itération, comme on est resté dans la boucle, c'est que $j < n-1$ et $v[j] < v[j+1]$, comme on sait que
$$\forall k \in [\![i, j-1]\!], v[k] < v[k+1]$$
on en déduit
$$\forall k \in [\![i, j]\!], v[k] < v[k+1]$$
donc $I(j+1)$ est vérifié ce qui correspond à la nouvelle valeur de j.

Ainsi, c'est bien un invariant et en sortie de boucle on a
$$\forall k \in [\![i, j]\!], v[k] < v[k+1]$$
En notant, $N = j-i$, on a alors
$$\forall k \in [\![i, i+N-1]\!], v[k] < v[k+1]$$
et,

- soit $j = n-1 \iff i+N+1=n$
- soit $v[j] = v[i+N] \geq v[i+N+1] = v[j+1]$.

Cela correspond donc à la spécification donnée par la question.

Q.8

En déduire une fonction de prototype

```
int plus_grande_poussee(int t[], int n);
```

qui renvoie la plus grande valeur de la poussée.
Attention votre fonction devra appeler la fonction précédente.

R.8

```
int plus_grande_poussee(int t[], int n) {
    int m = 0;
    for(int i = 0; i < n-1; i++) {
        int v = poussee(t, n, i);
        if (v > m) {
            m = v;
        }
    }
    return m;
}
```

Q.9

Prouvez la correction de la fonction précédente en supposant la fonction `poussee` correcte.

R.9

On remarque qu'il n'est pas question de terminaison vu qu'il s'agit d'une boucle inconditionnelle.
On va valider l'invariant $I(m,i) \cong \forall k \in [\![0, i-1]\!], m \geq \texttt{poussee}(t,n,k)$.

- Initialisation : au départ $m = 0$ et $i = 0$, donc la plage est vide et la propriété est vérifiée. Ainsi, $I(0,0)$ est vérifiée.
- Itération : Si $I(m,i)$ est vérifié en début d'itération, on a deux cas, soit $poussee(t,n,i) \leq m$ et m ne change pas, soit $poussee(t,n,i) > m$ et m devient $poussee(t,n,i)$. Dans tous les cas la valeur en fin d'itération de m est plus grande que tous les $poussee(t,n,k)$ pour $k \in [\![0,i]\!]$.

Ainsi, en sortie de boucle, on a $I(m,n)$ vérifié : m est plus grand que toutes les valeurs $pousee(t,n,k)$.

Remarque 2. ***Initialiser le maximum courant pour $i = 0$ ou avec une valeur par défaut ?***
Ici, on a fait le choix de l'initialiser à 0 car on sait que la poussée est dans $\mathbb{N}$. Mais quand on est dans $\mathbb{Z}$, comme c'est non minoré, il n'y a pas de bonne valeur pour initialiser, et dans ce cas, on place la valeur pour le premier indice comme valeur initiale.
Par contre, cela complique la preuve de correction, donc, quand on peut l'éviter, comme ici, on se contente d'initialiser avec un minimum absolu.

Q.10

Justifiez rapidement que si le tableau comporte n valeurs, alors la fonction précédente effectue de l'ordre de n^2 opérations élémentaires (affectations, accès mémoire, opérations arithmétiques ou de comparaison).

R.10

La fonction `poussee` effectue une boucle d'au plus $n - i$ itérations effectuant chacune des opérations élémentaires. Ainsi, chaque appel fait au maximum de l'ordre de n opérations élémentaires. Comme `plus_grande_poussee` effectue n appels à `poussee` on a de l'ordre n^2 opérations élémentaires.

Plutôt que la vitesse instantanée, on va s'intéresser maintenant à la vitesse moyenne sur une suite de mesures, ce qui revient à s'intéresser aux sommes de vitesses. On parle de *largeur de fenêtre* pour le nombre de mesure que l'on considère.
Ainsi, sur l'exemple précédent et avec une largeur de fenêtre de 3, la première somme vaut $199 + 200 + 208 = 607$ et la seconde 618 Il y avait 10 valeurs dans cet exemple, mais avec une fenêtre de largeur 3, on ne peut calculer que 8 sommes, on obtient ainsi la suite de sommes suivantes :

$$607 \quad 618 \quad 618 \quad 617 \quad 647 \quad 717 \quad 769 \quad 792$$

On peut alors considérer de même les accélerations entre ces sommes, moments où la somme augmente **strictement**. Pour l'exemple on a donc :

$$607 \nearrow 618 = 618 \searrow 617 \nearrow 647 \nearrow 717 \nearrow 769 \nearrow 792$$

Q.11

Écrire une fonction de prototype

```
int accelerations_fenetre(int v[], int n, int k);
```

qui prend en entrée un tableau de vitesses donné par v et n ainsi qu'une largeur de fenêtre k et renvoie le nombre d'accélérations associé.
Exemple : pour $k = 3$ et les vitesses de l'exemple précédent, votre fonction renverra la valeur 5.
Indication : il n'y a pas besoin de stocker les sommes de fenêtre dans un tableau, on peut tout faire uniquement avec un nombre fini de variables locales entières.

R.11

```
int accelerations_fenetre(int v[], int n, int k) {
    if (k+1 > n) // pas de fenetres
        return 0;
    int s = 0;
    for(int i = 0; i < k; i++) {
        s = s + v[i];
    }
    int accel = 0;
    for(int j = 0; j < n-k; j++) {
        int s2 = s - v[j] + v[j+k];
```

```
        if(s2 > s) {
            accel += 1;
        }
        s = s2;
    }
    return accel;
}
```

Q.12

Prouvez soigneusement la correction de votre fonction.

R.12

Il est facile de se tromper d'un cran dans la fonction précédente. Il y a un terme consacré pour ces problèmes : *off by one*. Ici, faire la preuve de correction permet d'être certain de ne pas s'être trompé.
La fonction est directement correct quand il y a `0` ou `1` fenêtre uniquement, car on renvoie `0` et qu'il y a bien `0` accéleration possible.
On remarque rapidement que la structure du compte est la même que dans la fonction `accelerations` donc, il suffit de vérifier que `s` contient bien la somme de la fenêtre.
On suppose assez rapidement que la valeur initiale $s = \sum_{i=0}^{k-1} v[i]$.
On va ainsi montrer que la propriété $I(j) \cong s = \sum_{i=j}^{j+k-1} v[i]$ est un invariant.
Avant la boucle, $j = 0$ et on a supposé $I(0)$. Si en début d'itérations, on a $I(j)$, alors $s = \sum_{i=j}^{j+k-1} v[i]$ donc $s2 = s - v[j] + v[j+k] = \sum_{i=j+1}^{j+k} v[i]$ et ainsi la valeur de s en fin d'itération vérifié bien $I(j+1)$.
En sortie de boucle, on a $I(n-k)$ vérifié donc $s = \sum_{i=n-k}^{n-1} v[i]$ est la dernière somme de fenêtre considérée et c'est bien la dernière fenêtre.

3 Planifier sa route - OCaml

Pour votre route, vous devrez passer d'île en île. Afin de passer à une île suivante, il est nécessaire d'attendre une certaine durée pour que votre *log2pos*, une sorte de boussole, puisse vous indiquer la direction de la prochaine île.
On vous a fourni une `int list` contenant les temps qu'il faut attendre pour chaque ile.

Q.13

Écrire une fonction de type :

```
temps_total : int list -> int
```

qui détermine le temps total qu'il faudra attendre.

R.13

```
let rec temps_total l =
    match l with
    | [] -> 0
    | t::q -> t + temps_total q
```

Q.14

Écrire une fonction de type :

```
plus_dattente : int list -> int
```

qui détermine l'indice dans la liste de l'ile où il faudra attendre le plus longtemps.

R.14

Cette fonction est plutôt délicate à écrire en récursif car on a besoin de beaucoup de variables temporaires qui deviennent des paramètres de fonctions auxiliaires.

```
let plus_dattente l =
```

```
    let rec aux l max_courant ind_max_courant indice_actuel =
        match l with
        | [] -> ind_max_courant
        | t::q ->
            if t > max_courant
            then aux q t indice_actuel (indice_actuel+1)
            else aux q max_courant ind_max_courant (indice_actuel+1)
    in match l with
       | [] -> failwith "Vide"
       | t::q -> aux q t 0 1
```

Une bonne manière de programmer une telle fonction est de penser à une version impérative et à transformer l'itération en une fonction : valeurs avant → valeur après.
Ainsi, en Python, on aurait pu écrire :

```
def plus_dattente(l):
    max_courant = t[0]
    ind_max_courant = 1

    for indice_actuel in range(1,len(l)):
        t = l[indice_actuel]
        if t > max_courant:
            max_courant = t
            ind_max_courant = indice_actuel

    return ind_max_courant
```

Il se trouve que le temps à attendre n'est qu'une approximation grossière de votre traversée car il faut prendre en compte le trajet entre chaque ile.
On considère alors le type suivant :

```
type donnee_nav =
    Trajet of int | Ile of int
```

Votre route est alors une liste alternant entre valeurs `Trajet` et valeurs `Ile`.

Q.15

Écrire une fonction de type :

`alernante : donnee_nav list -> bool`

qui indique si la liste alterne ainsi.

R.15

On peut écrire cette fonction de plusieurs manières. Avancer en regardant deux éléments est surement le plus simple.

```
let rec alternante l =
    match l with
    | [] | [_] -> true
    | Trajet _ :: Ile v :: q -> alternante (Ile v :: q)
    | Ile _ :: Trajet v :: q -> alternante (Trajet v :: q)
    | _ -> false
```

On pouvait aussi faire deux fonctions auxiliaires mutuellement récursives permettant de distinguer les cas où on commence avec une ile ou un trajet.

```
let alternante l =
    let rec ile_alt l =
        match l with
        | [] -> true
        | Trajet _ :: q -> false
        | Ile _ :: q -> trajet_alt q
    and trajet_alt l =
```

```
        match l with
        | [] -> true
        | Ile _ : q -> false
        | Trajet _ :: q -> ile_alt q
    in ile_alt l || trajet_alt l
```

Q.16

Écrire une fonction de type :

`ile_depart_fin : donnee_nav list -> bool`

qui indique si la liste commence et finit par une `Ile`.

R.16

```
let ile_depart_fin l =
    let dernier_ile l =
        match l with
        | [] -> false
        | [Ile _ ] -> true
        | _::q -> dernier_ile q in
    match l with
    | Ile _ :: q -> dernier_ile q
    | _ -> false
```

Q.17

Écrire une fonction de type :

`temps_total_nav : donnee_nav list -> int`

qui détermine le temps total, trajet et attente, de la route.

R.17

```
let rec temps_total_nav l =
    match l with
    | [] -> 0
    | Trajet t :: q | Ile t :: q ->
        t + temps_total_nav q
```

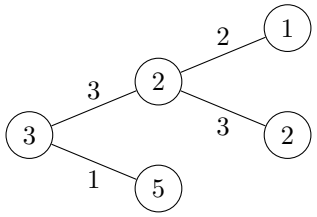
Il se trouve que votre route n'est pas forcément linéaire : il peut il y avoir des choix entre plusieurs îles.
On considère alors le type suivant :

```
type route =
    | Finie
    | Simple of donnee_nav * route
    | Double of (donnee_nav * route)
              * (donnes_nav * route)
```

Un couple du type `donnee_nav * route` représente ainsi un bout du parcours, soit d'une île, soit d'un trajet, et on continue la route. On a alors une route qui est soit finie, soit simple quand il n'y a pas de choix, soit double quand il on a le choix entre deux parcours.
Par exemple, la valeur :

```
Simple(Ile 3,
  Double(
    (Trajet 3,
        Simple(Ile 2,
            Double((Trajet 2, Simple(Ile 1, Finie)),
                (Trajet 3, Simple(Ile 2, Finie))))),
    (Trajet 1, Simple(Ile 5, Finie))))
```

représente la route suivante :



Q.18

Écrire une fonction de type

`temps_total_route : route -> int`

qui renvoie la somme des temps de l'ensemble de la route.

R.18

```
let rec temps_total_route l =
    match l with
    | Finie -> 0
    | Simple(Trajet t, q)
    | Simple(Ile t, q) ->
        t + temps_total_route q
    | Double(r1,r2) ->
        let aux c =
            match c with
            | (Ile t, r)
            | (Trajet t, r) -> t + temps_total_route r
        in aux r1 + aux r2
```

Q.19

Écrire une fonction de type

`alternant_route : route -> bool`

qui détermine si une route alterne toujours comme sur l'exemple, entre île et trajet.

R.19

Ici, la version avec récursivité croisée semble plus intéressante.

```
let alternant_route l =
    let rec alt_ile l =
        match l with
        | Finie -> true
        | Simple(Ile _, r) -> alt_trajet r
        | Double( (Ile _, r1), (Ile _, r2) ) ->
            alt_trajet r1 && alt_trajet r2
        | _ -> false
    and alt_trajet l =
        match l with
        | Finie -> true
        | Simple(Trajet _, r) -> alt_ile r
        | Double( (Trajet _, r1), (Trajet _, r2) ) ->
            alt_ile r1 && alt_ile r2
        | _ -> false
    in alt_ile l || alt_trajet l
```

Q.20

Écrire une fonction de type

`temps_min_route : route -> int`

qui renvoie le temps du chemin le plus court dans cette route.
Indication : sur l'exemple, on constate que le chemin le plus court est donné par le chemin du bas `Ile 3 - Trajet 1 - Ile 5` pour une durée de 9.

R.20

```
let rec temps_min_route l =
    match l with
    | Finie -> 0
    | Simple(Trajet t, q)
    | Simple(Ile t, q) ->
        t + temps_min_route q
    | Double(r1,r2) ->
        let aux c =
            match c with
            | (Ile t, r)
            | (Trajet t, r) -> t + temps_min_route r
        in min (aux r1) (aux r2)
```

Q.21

Écrire une fonction de type

`est_chemin : route -> donnee_nav list -> bool`

qui vérifie qu'un chemin donné sous la forme d'une liste de données de navigation est dans la route en partant du départ et en finissant complètement.
Indication : dans la route précédente, on peut constater que `[Ile 3; Trajet 3; Ile 2; Trajet 2; Ile 1]` est un chemin de la route, mais pas `[Ile 3; Trajet 1]` (pas finie) ou `[Ile 3; Trajet 4]` (données non présente).

R.21

```
let rec est_chemin r l =
    match r, l with
    | Finie, [] -> true
    | Simple(Ile a, r'), Ile b::q when a = b
        -> est_chemin r' q
    | Simple(Trajet a, r'), Trajet b::q when a = b
        -> est_chemin r' q
    | Double( (Ile a,r'), _ ), Ile b::q when a = b
        -> est_chemin r' q
    | Double( _, (Ile a,r') ), Ile b::q when a = b
        -> est_chemin r' q
    | Double( (Trajet a,r'), _ ), Trajet b::q when a = b
        -> est_chemin r' q
    | Double( _, (Trajet a,r') ), Trajet b::q when a = b
        -> est_chemin r' q
    | _ -> false
```

Note : en raison des `when` on ne peut pas factoriser les cas.

Q.22

Écrire une fonction de type

`chemins : route -> donnee_nav list list`

qui renvoie la liste de tous les chemins présents dans la route.
Indication : pour l'exemple, cette fonction va renvoyer :

```
[[Ile 3; Trajet 3; Ile 2; Trajet 2; Ile 1];
[Ile 3; Trajet 3; Ile 2; Trajet 3; Ile 2];
[Ile 3; Trajet 1; Ile 5]]
```

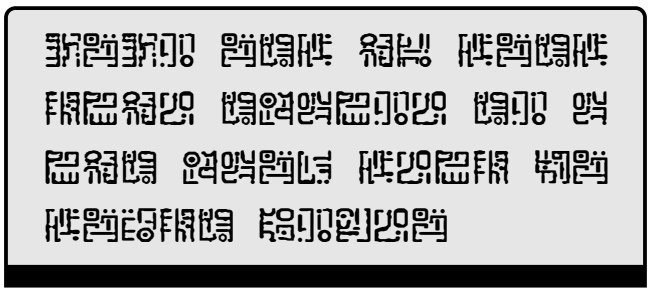
R.22

```
let rec ajoute x l =
    match l with
    | [] -> []
    | t::q -> (x::t) :: ajoute x q

let rec chemins r =
    match r with
    | Finie -> [ [] ]
    | Simple(v,r') -> ajoute v (chemins r')
    | Double((v1,r1),(v2,r2)) ->
        ajoute v1 (chemins r1)
        @ ajoute v2 (chemins r2)
```

4 Décoder les Codeglyphes

Tout au long de votre périple, votre exploratrice, Nicode, déchiffre des stèles contenant d'étranges caractères, les *Codeglyphes* :



Après avoir passé beaucoup de temps sur le déchiffrage, vous comprenez que tout repose sur votre interprétation de quatre Codéglyphes que l'on vous donne ici sous forme déchiffrée :

```
#include <stdio.h> int m
ain(void) {int t[]={1,1,
0,4,1,2,1,5,0,8,0,4,0,2,
0,4};for(int i=0;i<16;i=
i+2)printf("%c",'A'+10*t
[i]+t[i+1]);}
```

```
#include <stdio.h> int m
ain(void){int t[]={3,0,1
3,18,11,21,23,21,24,24,4
};int b=t[3]/2;int a=t[b
]-t[b+1];for (int i=0;i<
=b+1;i++){int b=t[i];if(
t[i]>a)b=t[t[i]-a];prin
tf("%c",'A'+b);}}
```

```
let rec f x xx=if x=0 th
en [] else x mod xx::f(x
/xx)xx and g ll l1 l=mat
ch ll with []->0|x::xx->
x*l1+g xx(l*l1) l in pri
nt_int(g(f 73 4)1 3)
```

```
#include <stdio.h> void d(ch
ar s[], int k){for(int i=0;i
<10;i++)s[i]='A'+(((s[i]-'A'
)+k)%26);}int main(void){cha
r s[]="MLDLCXZTCP";d(s,
;printf(s);}
```

Q.23

On vous demande, pour ces quatres programmes :

- de les présenter sous une forme lisible et acceptée par un compilateur. Cela peut passer par le renommage des variables ou fonctions.
- de dérouler leur exécution, attention pour le quatrième programme, vous devez devinez le bout de code non déchiffré.
- d'expliquer ce que fait le programme dans des termes simples.

Indication on rappelle que les caractères sont dans l'ordre alphabétique depuis `'A'`. Ainsi `'A'+3` est le caractère `'D'`.

R.23

1.
```c
#include <stdio.h>

int main(void) {
    int t[]={1,1,0,4,1,2,1,5,0,8,0,4,0,2,0,4};

    for(int i=0;i<16;i=i+2) {
        printf("%c",'A'+10*t[i]+t[i+1]);
    }
}
```

   On comprend ainsi que ce programme va lire le numéro dans l'alphabet, en commençant à $A = 0$ en décimal sur deux cases consécutives de t. Ainsi, les numéros de lettres sont $11, 4, 12, 15, 8, 4, 2, 4$ ce qui correspond à LEMPIECE.

2.
```c
#include <stdio.h>

int main(void) {
    int t[]={3,0,13,18,11,21,23,21,24,24,4};
    int b=t[3]/2; // 18/2=9
    int a=t[b]-t[b+1]; // t[9]-t[10]=24-4=20

    for (int i=0;i<=b+1;i++){
        int c=t[i]; // on change le nom
        if(t[i]>a)
            c=t[t[i]-a];
        printf("%c",'A'+c);
    }
}
```

Ici, le plus simple semble de faire un tableau de valeurs

i	$t[i]$	$t[i]-a$	$t[t[i]-a]$	c	$'A'+c$
0	3			3	D
1	0			0	A
2	13			13	N
3	18			18	S
4	11			11	L
5	21	1	0	0	A
6	23	3	18	18	S
7	21	1	0	0	A
8	24	4	11	11	L
9	24	4	11	11	L
10	4			4	E

et on lit donc DANSLASALLE.

3.
```
let rec f x xx =
    if x=0
    then []
    else x mod xx
        :: f (x/xx) xx
and g ll l1 l =
    match ll with
    []->0
    |x::xx-> x * l1 + g xx (l*l1) l
in print_int (g (f 73 4) 1 3)
```

qu'on peut rendre plus lisible en nommant les variables de manière raisonnable :

```
let rec decompose n base =
    if n=0
    then []
    else n mod base :: decompose (n/base) base
and compose ecriture acc base =
    match ecriture with
    []->0
    | digit::ecriture'-> digit * base
        + g ecriture' (base*acc) base
in print_int (compose (decompose 73 4) 1 3)
```

On voit alors qu'il s'agit de décomposer 73 en base 4 et d'en déduire le nombre en base 3 en lisant cette écriture. Ainsi 73 donne la liste `[1;2;0;1]` qui correspond en base 3 à $1+2\times 3+27=34$.

4.
```
#include <stdio.h>

void d(char s[], int k) {
    for(int i=0;i<10;i++)
        s[i]='A'+(((s[i]-'A')+k)%26);
}

int main(void){
    char s[]="MLDLCXZTCP";
    d(s,???);
    printf(s);
}
```

On comprend que la fonction `d` est un décalage vers la droite de k unités. Il suffit alors de chercher tous les décalages jusqu'à obtenir une valeur signifiante.

k	s
0	$MLDLCXZTCP$
1	$NMEMDYAUDQ$
2	$ONFNEZBVER$
3	$POGOFACWFS$
4	$QPHPGBDXGT$
5	$RQIQHCEYHU$
6	$SRJRIDFZIV$
7	$TSKSJEGAJW$
8	$UTLTKFHBKX$
9	$VUMULGICLY$
10	$WVNVMHJDMZ$
11	$XWOWNIKENA$
12	$YXPXOJLFOB$
13	$ZYQYPKMGPC$
14	$AZRZQLNHQD$
15	$BASARMOIRE$
16	$CBTBSNPJSF$
17	$DCUCTOQKTG$
18	$EDVDUPRLUH$
19	$FEWEVQSMVI$
20	$GFXFWRTNWJ$
21	$HGYGXSUOXK$
22	$IHZHYTVPYL$
23	$JIAIZUWQZM$
24	$KJBJAVXRAN$
25	$LKCKBWYSBO$

On pouvait donc lire BASARMOIRE.

5. Au final, la phrase cachée est LEMPIECEDANSLASALLE34BASARMOIRE.

Une fois les quatre programmes compris, vous avez pu découvrir l'emplacement du M-PIece ! Comme un vrai pirate, vous avez alors partagé le trésor avec tout l'équipage.