Informatique
MP2I 2022/2023 — DS2 — 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



Résumé des épisodes précédents : *À bord de votre bateau, le* ***Coding Merry****, vos coéquipiers et vous, Camel C. Civvy, êtes partis en quête du trésor légendaire : le M-PIece! Dans ce sujet, vous allez résoudre des problèmes pour y arriver. Et, qui sait, ce trésor se trouve peut-être au CIV.*

Ce problème est constitué de quatre parties indépendantes.

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 :

```c
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;
        }
    }
    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.

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 $0 \leq r < b$.

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 ↗ :

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

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.

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.

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.

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 ↗ 200 ↗ 208 ↗ 210 et 200 ↗ 207 ↗ 240 ↗ 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.

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.

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.

Q.9

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

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).

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'accélerations 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.

Q.12

Prouvez soigneusement la correction de votre fonction.

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.

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.

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.

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`.

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.

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 :

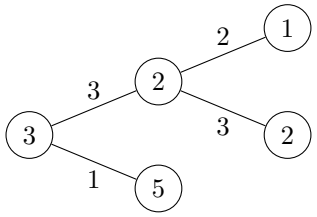
```
type route =
    | Finie
    | Simple of donnee_nav * route
    | Double of (donnee_nav * route)
             * (donnes_nav * route)
```

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(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.

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.

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.

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).

Q.22

Écrire une fonction de type

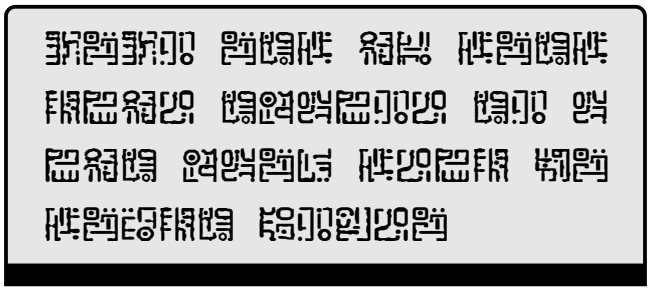
`chemins : route -> donnee_nav list list`

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]]
```

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'`.

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.