Devoir surveillé d'informatique

Consignes

- Les programmes demandés doivent être écrits en C, on suppose que les librairies standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`, `<stdassert.h>`, ...) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ Exercice 1 : *Questions de cours*

On considère l'algorithme suivant :

Algorithme : Multiplier sans utiliser `*`

Entrées : $n \in \mathbb{N}, m \in \mathbb{N}$

Sorties : nm

```
1 r ← 0
2 tant que m > 0 faire
3 |   m ← m − 1
4 |   r ← r + n
5 fin
6 return r
```

1. Donner les valeurs successives prises par les variables m, n et r si on fait fonctionner cet algorithme avec $n = 7$ et $m = 4$. On pourra recopier et compléter le tableau suivant :

	n	m	r
valeurs initiales	7	4	0
après un tour de boucle	...	...	...
après deux tours de boucle	...	...	...
après trois tours de boucle	...	...	...
après quatre tours de boucle	...	...	...

2. Donner une implémentation de cet algorithme en langage C sous la forme d'une fonction `multiplie` de signature `int multiplie(int n, int m)`. On précisera soigneusement la spécification de cette fonction en commentaire dans le code et on vérifiera les préconditions à l'aide d'instructions `assert`.
3. Donner la définition d'un variant de boucle, puis prouver que cet algorithme termine.
4. Donner la définition d'un invariant de boucle, puis prouver que cet algorithme est correct.

❑ Exercice 2 : *Retourner un tableau*

Dans cet exercice, on s'intéresse à un algorithme permettant de « retourner » un tableau c'est à dire réorganiser l'ordre de ses éléments de façon à ce que le premier élément devienne le dernier, le second devienne l'avant dernier et ainsi de suite. Par exemple, le tableau `{2, 7, 1, 9, 3}` deviendrait `{3, 9, 1, 7, 2}`. En notant, t_0 le tableau initial, et t_r le tableau « retourné » on a donc pour tout $k \in [\![0; n-1]\!]$, $t_r[k] = t_0[n-1-k]$. On propose pour cela l'algorithme suivant :

Algorithme : Retourner un tableau

Entrées : Un tableau t de taille n

Sorties : Aucune

Résultat : Le tableau t est modifié (retourné)

```
1 i ← 0
2 j ← n − 1
3 tant que j − i > 0 faire
4 |   échanger les éléments d'indice i et j dans t
5 |   i ← i + 1
6 |   j ← j − 1
7 fin
```

1. Montrer que cet algorithme termine.
2. Montrer que la propriété suivante notée I est un invariant de l'algorithme : $i + j = n - 1$.
3. Montrer que cet algorithme est correct, on pourra considérer l'invariant I' : « *Le tableau t contient les valeurs de t_r (le tableau retourné) pour tous les indices $k \in [\![0; i-1]\!] \bigcup [\![j+1; n-1]\!]$* » et utiliser l'invariant I de la question précédente.
4. On souhaite utiliser cet algorithme afin d'écrire en langage C une fonction `retourner_str` qui retourne une chaine de caractères, c'est à dire que par exemple, la chaine `"MP2I"` devient `"I2PM"`. Rappeler la façon dont sont implémentées en C les chaines de caractères et expliquer pour quelle raison il n'est pas utile de fournir la longueur de la chaine en paramètre à la fonction `retourner_str`.
5. Ecrire une fonction de signature `void echange(char s[], int i, int j)` qui échange dans `s` les caractères situés aux indices `i` et `j`.
6. Ecrire une implémentation de la fonction `retourner_str` sans utiliser les fonctions de la librairie `<string.h>`.
7. On rappelle qu'un palindrome est un mot qui se lit de la même façon de droite à gauche ou de gauche à droite, par exemple « *radar* » est un palindrome, mais « *tata* » n'en est pas un. Pour tester si un mot est un palindrome un élève propose la solution suivante :

```
1  bool palindrome(char s[])
2  {
3      //renvoie true ssi s est un palindrome
4      char copie[] = s;
5      retourner_str(copie);
6      return s==copie;
7  }
```

   Ce programme ne compile pas et produit une erreur à la ligne 4 : « *error : invalid initializer* ». Indiquer la source de cette erreur et expliquer comment la corriger, *on ne demande pas d'écrire une fonction corrigée de la fonction palindrome* mais d'indiquer de façon succinte la source de l'erreur commise à la ligne 4.
8. On suppose corrigée l'erreur commise à la ligne 4, cette fonction est-elle correcte ? Justifier votre réponse.

❑ **Exercice 3** : *Lecture et compréhension d'un code C*

On considère la fonction `mystere` suivante :

```
1  int mystere(int n)
2  {
3      assert(n > 1);
4      int d = 2;
5      while (n % d != 0)
6      {
7          d = d + 1;
8      }
9      return d;
10 }
```

1. Quelle est la valeurs renvoyée par l'appel `mystere(35)` ? et par `mystere(13)` ?
2. Quel sera le résultat de l'exécution d'un programme effectuant l'appel `mystere(1)` ?
3. Proposer une spécification aussi précise que possible pour cette fonction.
4. Prouver la terminaison de cette fonction.
5. En utilisant la fonction précédente, écrire une fonction `est_premier` de prototype :
   `bool est_premier(int n)` qui prend en entrée un entier $n \in \mathbb{N}$ et qui renvoie `true` si et seulement si `n` est premier.

❑ **Exercice 4** : *Recherche de minimums*

On donne ci-dessous le programme d'un élève en C afin de rechercher le minimum d'un tableau d'entiers :

```c
int min_eleve(int tab[], int taille)
{
    // Renvoie le minimum des éléments de tab (supposé non vide)
    assert(taille > 0);
    int cmin = 0;
    for (int i = 0; i < taille; i++)
    {
        if (tab[i] < cmin)
        {
            cmin = tab[i];
        }
    }
    return cmin;
}
```

1. Proposer un test montrant que cette fonction ne répond pas à sa spécification.
2. Corriger cette fonction afin de la rendre conforme à sa spécification.

On s'intéresse maintenant à la recherche des deux plus petites valeurs d'un tableau contenant au moins deux éléments. Pour cela on initialise deux valeurs `min1` et `min2` aux deux premières valeurs du tableau avec `min1 <= min2`, puis on parcourt le reste du tableau en mettant à jour ces valeurs en fonction de la valeur `tab[i]` rencontrée dans le tableau.

3. Expliquer succintement, comment mettre à jour `min1` et `min2` de façon à préserver l'invariant suivant : `min1` et `min2` sont les deux plus petites du sous tableau `tab[0] ... tab[i-1]` et `min1 <= min2` (on pourra distinguer les cas où `tab[i]` est inférieur à `min1` ou compris entre `min1` et `min2`).
4. On propose pour implémenter cette fonction en C, d'utiliser un type structuré `couple` contenant deux valeurs de type `int`. Donner la définition de ce type structuré qu'on appellera `couple` et dont les champs seront appeles `premier` et `second`.
5. Ecrire la fonction de signature `couple deuxmin(int tab[], int n)` qui prend en argument un tableau et sa taille et renvoie ses deux plus petites valeurs dans une variable de type `couple`.
6. Une autre possibilité d'implémentation consiste à passer en paramètre deux pointeurs vers des entiers qui seront modifiés dans la fonction et à ne rien renvoyer. Donner alors la signature de la fonction ainsi que son implémentation.