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

```
r ← 0
tant que m > 0 faire
    m ← m − 1
    r ← r + n
fin
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	7	3	7
après deux tours de boucle	7	2	14
après trois tours de boucle	7	1	21
après quatre tours de boucle	7	0	28

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`.

```c
// Prends en entrée deux entiers positifs n et m et renvoie leur produit nm
int multiplie(int n, int m)
{
    assert(n >= 0 && m >= 0);
    int r = 0;
    while (m > 0)
    {
        m = m - 1;
        r = r + n;
    }
    return r;
}
```

3. Donner la définition d'un variant de boucle, puis prouver que cet algorithme termine.

Un *variant de boucle* est une quantité qui dépend des variables du programmes et est :

1. entière,
2. positive,
3. strictement décroissante.

. Dans l'algorithme ci-dessus, la quantité m est un variant de boucle, en effet :

1. $m \in \mathbb{N}$ par précondition.
2. $m \in \mathbb{N}$ par précondition puis m reste positif car par condition d'entrée dans la boucle $m \geqslant 1$ et dans la boucle on décrémente m donc après un passage dans la boucle m reste positif ou nul.
3. m décroit strictement car m est diminué de 1 lors de chaque passage dans la boucle.

L'algorithme termine car on a trouvé un variant de boucle.

4. Donner la définition d'un invariant de boucle, puis prouver que cet algorithme est correct.

Un *invariant de boucle* est une propriété qui dépend des variables du programme et qui est :

1. vraie avant d'entrer dans la boucle (initiatilisation)
2. reste vraie après un tour de boucle si elle l'était au tour précédent (conservation)

En sortie de boucle, la validité d'un invariant permet de prouver la correction de l'algorithme.
On note, m_0 la valeur initiale de m, montrons que la propriété I : « $r = (m - m_0)n$ » est un invariant de boucle.

1. Avant d'entrée dans la boucle $m = m_0$ donc $(m - m_0)n = 0$ et comme r est initialisé à 0 la propriété I est vérifiée.
2. On suppose I vérifié à l'entrée de la boucle et on note r' (resp. m') les valeurs prises par r (resp. m) au tour de boucle suivant, alors :
   $(m' - m_0)n = (m + 1 - m_0)n$, or I étant vérifié à l'entrée de boucle $(m - m_0)n = r$ donc
   $(m' - m_0)n = r + n$ et comme $r' = r + n$
   $(m' - m_0)n = r'$ et donc I est vérifiée.

En sortie de boucle, puisque $m = 0$, cette invariant prouve que $r = m_0 n$ et donc l'algorithme est correcte.

❑ **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.

Dans le cas où le tableau est vide, l'algorithme termine directement sans entrer dans la boucle `while`. Sinon,

- $j - i$ est entier comme différence de deux entiers
- $j - i$ est positif avant d'entrer dans la boucle ($n \geqslant 1$) et reste positif par condition d'entrée dans la boucle
- $j - i$ décroît strictement puisque i est incrémenté et j décrémenté

2. Montrer que la propriété suivante note I est un invariant de l'algorithme : $i + j = n - 1$.

- Avant d'entrer dans la boucle $i = 0$ et $j = n - 1$ et donc I est vrai.
- On suppose I vraie en entrant dans la boucle, montrant qu'alors I est conservé lors d'un passage dans cette boucle, $i' = i + 1$ et $j' = j - 1$ sont après ce passage les nouvelles valeurs de i et j, on a donc $i' + j' = i + j$ or par hypothèse $i + j = n - 1$ donc I est conservée.

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.

On vérifie que l'invariant proposé est vraie :

- Avant d'entrer dans la boucle $[\![0; i-1]\!] \bigcup [\![j+1; n-1]\!] = \varnothing$ et donc I' est vraie par vacuité.
- Conservé par un passage dans la boucle. Si I' est vrai en entrant dans la boucle, alors on échange $t[i]$ avec $t[j]$ et comme d'après l'invariant I démontré à la question précédente $j = n - 1 - i$, on echange $t[i]$ avec $t[n - 1 - i]$. Comme par hypothèse les éléments situés avant i et après j sont déjà correctement échangé (I' supposé vraie) dans le nouveau tableau obtenu I' est vrai.

En sortie de boucle, on a $j - i \leqslant 0$ et donc $[\![0; i-1]\!] \bigcup [\![j+1; n-1]\!] = [\![0; n-1]\!]$, donc le tableau t contient bien les valeurs du tableau retourné.

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`.

En C, les chaines de caractères sont des tableaux d'entiers se terminant par le caractère sentinelle `'\0'`, on peut donc obtenir la taille d'une chaine de caractère en recherchant la position de ce caractère, il n'est pas utile de la passer en paramètre.

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`.

```c
void echange(char s[], int i, int j)
{
    // echange les caractères situés aux indices i et j dans s
    char temp = s[i];
    s[i] = s[j];
    s[j] = temp;
}
```

6. Ecrire une implémentation de la fonction `retourner_str` sans utiliser les fonctions de la librairie `<string.h>`.

```c
void retourner_str(char s[])
{
    // Retourne en place la chaine de caractère s
    int n = 0;
    while (s[n] != '\0')
    {
        n++;
    }
    int i = 0;
    int j = n - 1;
    while (j - i > 0)
    {
        echange(s, i, j);
        i = i + 1;
        j = j - 1;
    }
}
```

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 :

```c
bool palindrome(char s[])
{
    //renvoie true ssi s est un palindrome
    char copie[] = s;
    return (retourner_str(copie)==s);
}
```

Ce programme ne compile pas et produit une erreur à la ligne 4 : « *error : invalid initializer* » et aussi à la ligne 5. Indiquer la source de ces erreurs et expliquer comment les corriger. *on ne demande pas d'écrire une fonction corrigée de la fonction palindrome* mais d'indiquer de façon succinte la source des erreurs qu'elle contient.

> On peut pas en C affecter directement un tableau, on doit travailler élément par élément, on devrait donc utiliser une boucle `for` afin de copier un à un les caractères de s dans la chaine `copie`. De la même façon, pour tester l'égalité il faut procéder caractère par caractère dans une boucle.

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

On considère la fonction `mystere` suivante :

```c
int mystere(int n)
{
    assert(n > 1);
    int d = 2;
    while (n % d != 0)
    {
        d = d + 1;
    }
    return d;
}
```

1. Quelle est la valeurs renvoyée par l'appel `mystere(35)` ? et par `mystere(13)` ?

   `mystere(35)` renvoie 5 et `mystere(13)` renvoie 13.

2. Quel sera le résultat de l'exécution d'un programme effectuant l'appel `mystere(1)` ?

   Le programme s'arrête sur une erreur d'assertion à la ligne 3.

3. Proposer une spécification aussi précise que possible pour cette fonction.

   On peut propose la spécification suivante : « *Prend en entrée un entier n>1 et renvoie son premier diviseur strictement plus grand que 1* ».

4. Prouver la terminaison de cette fonction.

   Montrons que $n - d$ est un variant de boucle :
   - $n - d \in \mathbb{N}$ car n et d sont des entiers.
   - $n - d \geqslant 0$, en effet cela est vrai à l'initialisation ($d = 2$ et $n > 1$) et reste vrai à chaque passage dans la boucle car comme n divise n, par condition d'entrée dans la boucle $d < n$.
   - $n - d$ est strictement croissante car d est incrémenté à chaque tour de boucle.

   Donc cet algorithme termine.

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.

```
// Renvoie true ssi n est premier
bool est_premier(int n)
{
    assert(n >= 0);
    if (n == 0 || n== 1)
    {
        return false;
    }
    return (mystere(n) == n);
}
```

❑ **Exercice 4** : *Second minimum*

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

```
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.

   Si le tableau ne contient que des valeurs *strictement positives* alors le minimum étant initialisé à 0 le test de la ligne 8 n'est jamais vrai et donc la valeur renvoyée pour le minimum sera 0. On peut par exemple prendre le tableau `{2, 7, 5 }` son minimum est 2 et la fonction renverra `0`.

2. Corriger cette fonction afin de la rendre conforme à sa spécification.

```
1  int min_corrige(int tab[], int taille)
2  {
3      // Renvoie le minimum des éléments de tab (supposé non vide)
4      assert(taille > 0);
5      int cmin = tab[0];
6      for (int i = 1; i < taille; i++)
7      {
8          if (tab[i] < cmin)
9          {
10             cmin = tab[i];
11         }
12     }
13     return cmin;
14 }
```

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`).

   — Si `tab[i]<min1` alors `min2` prend la valeur de `min1` et `min1` celle de `tab[i]`
   — Si `min1<=v<min2` alors `min2` prend la valeur de `tab[i]`
   — Sinon `tab[i]` n'est pas l'un des deux minimums, on ne fait rien.

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`.

```
1  struct couple_s
2  {
3      int premier;
4      int second;
5  };
6  typedef struct couple_s couple;
```

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`.

```c
couple deuxmin(int tab[], int taille)
{
    assert(taille > 1);
    couple min;
    if (tab[0] < tab[1])
    {
        min.premier = tab[0];
        min.second = tab[1];
    }
    else
    {
        min.premier = tab[1];
        min.second = tab[0];
    }
    for (int i = 2; i < taille; i++)
    {
        if (tab[i] < min.premier)
        {
            min.second = min.premier;
            min.premier = tab[i];
        }
        else if (tab[i] < min.second)
        {
            min.second = tab[i];
        }
    }
    return min;
}
```

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.

```c
void deuxmin2(int tab[], int taille, int* min1, int* min2)
{
    assert(taille > 1);
    if (tab[0] < tab[1])
    {
        *min1 = tab[0];
        *min2 = tab[1];
    }
    else
    {
        *min1 = tab[1];
        *min2 = tab[0];
    }
    for (int i = 2; i < taille; i++)
    {
        if (tab[i] < *min1)
        {
            *min2 = *min1;
            *min1 = tab[i];
        }
        else if (tab[i] < *min2)
        {
            *min2 = tab[i];
        }
    }
}
```