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 : *Puissance*

1. Ecrire une fonction `valeur_absolue` qui prend en argument un entier n et renvoie sa valeur absolue $|n|$.
   On rappelle que : $|n| = \begin{cases} -n & \text{si } n < 0 \\ n & \text{sinon} \end{cases}$
2. Ecrire une fonction `puissance` qui prend en argument un flottant (type `double`) a et un entier n et renvoie a^n. On rappelle que pour $a \in \mathbb{R}^*$, $n \in \mathbb{Z}$:
   $$\begin{cases} a^n = \underbrace{a \times \cdots \times a}_{n \text{ facteurs}} & \text{si } n > 0, \\ a^0 = 1, & \\ a^n = \dfrac{1}{a^{-n}} & \text{si } n < 0. \end{cases}$$
   D'autre part $0^0 = 1$, $0^n = 0$ si $n \geqslant 0$ et les puissances négatives de zéro ne sont pas définies. On vérifiera la précondition $n \geqslant 0$ lorsque $a = 0$ à l'aide d'une instruction `assert`.
3. Tracer le graphe de flot de contrôle de cette fonction.
4. Proposer un jeu de test permettant de couvrir tous les arcs.

❑ **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)` ?
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** : *Programmation en C : algorithme de Luhn*
Un algorithme (appelé algorithme de Luhn, d'après le nom de son inventeur), est utilisé pour vérifier qu'un numéro de carte de crédit est valide, cela permet d'indiquer à un utilisateur une éventuelle erreur de saisie. Le but de l'exercice est de programmer cet algorithme en C, on prendra soin de préciser dans le code sous forme de commentaire les spécifications des fonctions demandées.

1. Ecrire une fonction `mult2` qui prend en entrée un entier naturel c compris entre 0 (inclus) et 9 (inclus), et renvoie $2c$ si $0 \leqslant 2c \leqslant 9$ et la somme des deux chiffres de $2c$ sinon. Par exemples :
   — `mult2(3)` renvoie 6 (car $2 \times 3 = 6$),
   — `mult2(7)` renvoie 5 (comme $2 \times 7 = 14$ on additionne $1 + 4$ et donc on renvoie 9),
   — `mult2(9)` renvoie 9 (comme $2 \times 9 = 18$ on additionne $1 + 8$ et donc on renvoie 9),
2. Pour vérifier que la fonction `mult2` est totalement correcte, dix tests suffisent, lesquels et pourquoi ? Donner ces dix tests sous forme d'instructions `assert`.
3. L'algorithme de Luhn consiste à faire la somme des chiffres du numéro de carte de crédit en utilisant au préalable la fonction `mult2` ci-dessus sur les chiffres de rang pair (c'est à dire en partant de la fin du nombre, le 2e chiffre, le 4e chiffre, ...). Si la somme obtenue est divisible par 10 alors le numéro est valide. Par exemple :
   — pour `267` on doit faire `2 + mult2(6) + 7` ce qui donne `2+3+7 = 12` et donc ce numéro est invalide.
   — pour `15782`, on doit faire la somme `1 + mult2(5) + 7 + mult2(8) + 2`, ce qui donne : `1+1+7+7+2 = 18` et donc ce numéro est invalide.
   — pour `124586`, on doit faire la somme `mult2(1) + 2 + mult2(4) + 5 + mult2(8) + 6`, ce qui donne : `2+2+8+5+7+6 = 30` et donc ce numéro est valide puisque `30` est divisible par `10`.
   a) Vérifier que le numéro `4762` est valide.
   b) Ecrire une fonction de prototype `bool valide(int n)` qui prend en entrée un numéro de carte de crédit et renvoie un booléen indiquant si ce numéro est valide.
   c) Ecrire une fonction qui prend en entrée un entier `n` et détermine quel chiffre ajouter à droite de ce nombre de façon à ce que le nombre ainsi formé soit un numéro de carte de crédit valide. Par exemple pour `543`, cette fonction renvoie le chiffre `9` car `5439` est un numéro de carte de crédit valide (`mult2(5) + 4 + mult2(3) + 9 = 1 + 4 + 6 + 9 = 20`) .