Correction des exercices de la session 3

1. (a) Pour résoudre ce problème, nous pouvons juste regarder les diviseurs de n en commençant par 1. Dès qu'on trouve le premier diviseur $d > 1$, on sait que $\frac{n}{d}$ est le plus grand diviseur de n. Afin de garder une complexité raisonnable, nous n'allons pas parcourir tous les diviseurs, nous allons nous arrêter à $\sqrt{n}$.

```
unsigned pgd(unsigned n) {
    unsigned d;
    unsigned res = 1;
    for (d = 1; (d*d <= n)|| (res != 1); d++) {
        if (!(n%d))
            res = n/d;
    }
    return res;
}
```

   (b) Dans le pire des cas, qui est si n est un nombre premier ou si n n'a qu'un seul diviseur autre que lui-même et 1 (e.g. 9), nous effectuerons $\sqrt{n}$ itérations avec des instructions à temps constant. Notre complexité en temps théorique est donc $\mathcal{O}(\sqrt{n})$.

2. (a) Pour résoudre ce problème, nous allons parcourir la chaine de caractères jusqu'à trouvé un caractère de terminaison et incrémenter un compteur. Nous avons cependant besoin de d'abord passer tous les caractères blancs jusqu'au premier qui ne l'est pas.

```
unsigned nb_char(char *c) {
    while (*c == ' ') {
        c++;
    }
    unsigned res = 0;
    while (*c) {
        res++;
        c++;
    }
    return res;
}
```

   (b) Pour montrer que la fonction est correcte, on doit montrer que loresque la fonction termine son exécution, la valeur qu'elle retourne est correcte. Comme nous avons 2 boucles séparées, nous allons réaliser l'invariant pour les 2.

      i. Nous devons d'abord montrer que les valeurs des variables après la première boucle sont correctes. Nous avons alors le triplet :

$$\{c = c_0\}$$

```
while(*c == ' ') {
    c++;
}
```

$$\{c = \text{ le premier élément qui n'est pas un caractère blanc}\}$$

      Pour trouver un invariant de boucle I, on caractérise le traitement effectué par la boucle jusqu'à une itération donnée. De plus, I doit être impliqué par la précondition, doit être

vrai autant avant qu'après une itération de la boucle *while*, et doit impliquer la postcondition après la dernière itération. Un invariant possible est :

$$I : \text{Tous les éléments entre } c_0 \text{ et } c \text{ non-compris sont des caractères blancs}$$

Montrons maintenant que cet invariant est valide :

- Initialement, on a $c = c_0$. Il n'y a aucun élément entre c_0 et c_0 non-compris. Il y a en effet aucun caractère blanc dans une chaine vide.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, c == \text{' '}\}$$

```
c++;
```

$$\{I\}$$

Montrons que ce triplet est valide en notant respectivement c et c' la valeur de c avant et après une certaine itération. Dans tous les cas, nous avons que $c' = c + 1$ ce qui signifie que nous passons à l'élément suivant. On a bien que, comme on réalise une itération, on a trouvé un nouveau caractère blanc, comme tous les éléments entre c_0 et c non-compris étaient des caractères blancs, on a que tous les éléments entre c_0 et c compris (correspond à $c + 1$ non-compris) sont des caractères blancs.

- En fin de boucle, on a I et $c \neq$ ' '. On a bien que tous les éléments avant celui-ci sont des caractères blancs, donc tous les éléments entre c_0 et c non-compris.

ii. Nous devons ensuite montrer que les valeurs des variables à la fin de la deuxième bocule sont correctes. Nous avons alors le triplet :

$$\{c = c_1, *c \neq \text{' '}, res = 0\}$$

```
while (*c) {
    res++;
    c++;
}
```

$$\{res = \text{ nombre de caractères entre } c_1 \text{ et la fin de la chaine}\}$$

Pour trouver un invariant de boucle I, on caractérise le traitement effectué par la boucle jusqu'à une itération donnée. De plus, I doit être impliqué par la précondition, doit être vrai autant avant qu'après une itération de la boucle *while*, et doit impliquer la postcondition après la dernière itération. Un invariant possible est :

$$I : res = \text{ nombre d'éléments entre } c_1 \text{ et } c \text{ non-compris}$$

Montrons maintenant que cet invariant est valide :

- Initialement, on a $c = c_1$. Il n'y a aucun élément entre c_1 et c_1 non-compris, on a bien $res = 0$.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, c \neq 0\}$$

```
res++;
c++;
```

$$\{I\}$$

Montrons que ce triplet est valide en notant respectivement c et c', et res et res' la valeur de c et res avant et après une certaine itération. Dans tous les cas, nous avons que $c' = c + 1$ ce qui signifie que nous passons à l'élément suivant. Nous avons également $res' = res + 1$. Comme res = nombre d'éléments entre c_1 et c non-compris, on a que res' = nombre d'éléments entre c_1 et c non-compris +1, comme nous avons trouvé un nouveau caractère qui n'est pas le caractère de terminaison, et donc res' = nombre de caractères entre c_1 et c compris, ce qui correspond au nombre de caractères entre c_1 et c' non-compris.

- En fin de boucle, on a I et $c = 0$. On a bien que res = nombre de caractères entre c_1 et c non-compris, res est bien égal au nombre de caractères à partir de c_1 sans compter le caractère de terminaison.

3. (a) Une façon de résoudre ce problème consiste à diviser répétitivement n par f tant que cela est possible tout en incrémentant un compteur à chaque fois.

```
unsigned mult_fact(unsigned n, unsigned f) {
    unsigned m;
    if (f == 1)
        return 0;
    for (m = 0; !(n%f); n/=f, m++);
    return m;
}
```

(b) Pour montrer que la fonction est correcte, on doit montrer que lorsque la fonction termine son exécution, la valeur qu'elle retourne est correcte. On souhaite alors établir la validité du triplet :

$$\{n > 0, f > 1\}$$

```
for (m = 0; !(n%f); n/=f, m++);
```

$$\{m = \text{ multiplicité de } f \text{ dans } n\}$$

En décomposant la boucle *for* en boucle *while*, on obtient :

$$\{n > 0, f > 1, m = 0\}$$

```
while (!(n%f)) {
    n /= f;
    m++;
}
```

$$\{m = \text{ multiplicité de } f \text{ dans } n\}$$

Pour trouver un invariant de boucle I, on caractérise le traitement effectué par la boucle jusqu'à une itération donnée. De plus, I doit être impliqué par la précondition, doit être vrai autant avant qu'après une itération de la boucle *while*, et doit impliquer la postcondition après la dernière itération. Un invariant possible est :

$$I : n = \frac{n_0}{f^m}$$

En d'autres termes, la valeur actuelle de n vaut la valeur initiale de n divisée par f^m.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $m = 0$. Comme $f^0 = 1$, pour tout $f > 1$, on a bien que $n = \frac{n_0}{1} = n_0$.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, !(n\%f)\}$$

```
n /= f;
m++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement m et m', et n et n' la valeur de la variable m et n avant et après l'itération concernée. À chaque itération, nous avons $n' = n/f$ et $m' = m + 1$. On a bien que, comme $n = \frac{n_0}{f^m}$, $n' = \frac{n_0}{f^m}/f = \frac{n_0}{f^{m+1}} = \frac{n_0}{f^{m'}}$

- En sortie de boucle, on a I et n n'est pas divisible par f qui implique bien que $m =$ multiplicité de f dans $n = n_0$, car si $n = \frac{n_0}{f^m}$ et n n'est pas divisible à nouveau par f, f^m est la plus grande puissance de f qui divise n_0.

4. (a) Pour résoudre ce problème, nous devons tout d'abord initialiser un maximum (comme ce sont des entiers non-signés, on sait que la plus petite valeur est 0) et regarder les éléments du tableau d'indice i et $i + 1$ pour voir s'ils sont égaux. Si c'est le cas, on compare avec le maximum, si le maximum est plus petit on change le maximum.

```
unsigned f(unsigned *t, unsigned n) {
    unsigned max = 0;
    for (unsigned i = 0; i < n - 1; i++) {
        if (t[i] == t[i+1]){
            if (max < t[i])
                max = t[i];
        }
    }
    return max;
}
```

   (b) Nous devons parcourir tout le tableau 1 fois, ce qui correspond à $n-1$ itérations. Toutes les autres instructions se font en temps constant. Nous avons donc une complexité en temps théorique de $\mathcal{O}(n)$.