Correction des exercices de la session 2

1. Voici un code correspondant à la fonction demandée :

```
unsigned palindrome(char * c){
    unsigned last;
    for(last = 0; c[last+1] != '/0'; last++);
    for(unsigned i = 0; i < last-i; i++){
        if (c[i] != c[last-i])
            return 0;
    }
    return 1;
}
```

   - Complexité : Nous avons deux boucles. La première va itérer n fois, où n correspond à la taille de la chaine de caractères. La deuxième boucle va itérer, au maximum, $n/2$ fois. Toutes les autres instructions se font en temps constant. Comme la deuxième boucle est après la première, et non pas à l'intérieur de celle-ci, nous avons une complexité en temps valant $\mathcal{O}(n) + \mathcal{O}(n)$ ce qui est approximé par $\mathcal{O}(n)$, où n correspond à la taille de la chaine de caractères.
   - Invariant : 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 :

$$\{c = c_0, \ldots, c_\ell (\text{où } \ell \text{ correspond à la taille de la chaine de caractères}), last = \ell - 1\}$$

```
for(unsigned i = 0; i < last-i; i++){
    if (c[i] != c[last-i])
        return 0;
}
```

$$\{1 \text{ si } \texttt{c} \text{ est un palindrome, 0 sinon}\}$$

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

$$\{c = c_0, \ldots, c_\ell (\text{où } \ell \text{ correspond à la taille de la chaine de caractères}), last = \ell - 1, i = 0\}$$

```
while(i < last-i){
    if (c[i] != c[last-i])
        return 0;
    i++;
}
```

$$\{1 \text{ si } \texttt{c} \text{ est un palindrome, 0 sinon}\}$$

     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 :

$$\begin{aligned} I : & 0 \leq i \leq \lceil last/2 \rceil \\ & \text{et } \forall 0 \leq j < i, c[j] = c[last - j] \end{aligned}$$

     En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, les éléments précédemment pointés par $c[i]$ et $c[last - i]$ sont identiques.

     Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = 0$ et $last = \ell - 1$, et donc une chaîne de caractères vide est bien un palindrome.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, i < last - i\}$$

```
if (c[i] != c[last-i])
    return 0;
i++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement i et i' la valeur de la variable i avant et après l'itération concernée.

  * Dans tous les cas, on a $i' = i+1$. Comme $0 \leq i < \lceil last/2 \rceil$, on a bien que $0 < i' \leq \lceil last/2 \rceil$.
  * Si $c[i] \neq c[last - i]$, alors l'instruction située dans le `if` s'exécute, et la boucle se termine. On a bien que $\forall 0 \leq j < i, c[j] = c[last - j]$.
  * Si $c[i] = c[last - i]$, alors l'instruction située dans le `if` ne s'exécute pas, et on a $c[i] = c[last]$, ce qui correspond à $c[i' - 1] = c[last - i' + 1]$. On garde bien alors que $\forall 0 \leq j < i + 1 c[j] = c[last - j]$, ce qui équivaut à $\forall 0 \leq j < i' c[j] = c[last - j]$.

- On peut sortir de la boucle de deux façons différentes. La première correspond à $\{I \wedge \neg cond\}$. On a alors que $i = \lceil last/2 \rceil$. Ce qui implique bien que la chaîne de caractères commençant par l'endroit pointé par `c` et se terminant au premier caractère terminateur forme un palindrome. Le deuxième cas est quand nous avons $\{I \wedge c[i] \neq c[last - i]\}$. Dans ce cas, nous avons bien que $i \leq \lceil last/2 \rceil$ mais également que tous les caractères précédents pouvaient former un palindrome : $\forall 0 \leq j < i c[j] = c[last - j]$.
- Il reste à démontrer que l'exécution de la fonction se termine toujours. Il faut donc trouver un variant de boucle qui est un entier non négatif dont la valeur décroît strictement à chaque itération de la boucle :

$$v = \lceil last/2 \rceil - i$$

2. Voici un code correspondant à la fonction demandée :

```
int f (int * t, unsigned l, int n){
    int d = 0;
    int f = l-1;
    int m = f/2;
    while(d <= f){
        if (t[m] == n)
            return m;
        if (t[m] > n)
            f = m-1;
        else
            d = m+1;
        m = (d+f)/2;
    }
    return -1;
}
```

- Complexité : À chaque itération de la boucle, nous allons diviser la taille du sous-tableau actuel par 2. Le nombre d'itération maximum sera donc de $\log_2 \ell$. La complexité en temps sera donc approximée par $\mathcal{O}(\log \ell)$.

- Invariant : 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 :

$$\{t = t_0, \ldots, t_\ell, \ell > 0, f = \ell - 1, m = f/2, d = 0\}$$

```
while(d <= f){
    if (t[m] == n)
        return m;
    if (t[m] > n)
        f = m-1;
    else
        d = m+1;
    m = (d+f)/2;
}
```

$$\{-1 \text{ si } n \notin t, \text{ l'indice de n dans t sinon}\}$$

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 :

$$\begin{aligned} I :& 0 \leq d \leq f + 1 \\ & \text{et } d - 1 \leq f \leq \ell - 1 \\ & \text{et } m = (f + d)/2 \end{aligned}$$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, d et f désigne les indices extrêmes d'un sous-tableau de t et que m désigne toujours l'indice au milieu du sous-tableau.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $d = 0$, $f = \ell - 1$ et $m = f/2$, on a bien que le sous-tableau de t actuel est le tableau en entier, dont les indices extrêmes sont 0 et $\ell - 1$ et que l'indice du milieu correspond bien à $0 + \ell - 1/2$.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, d \leq f\}$$

```
if (t[m] == n)
    return m;
if (t[m] > n)
    f = m-1;
else
    d = m+1;
m = (d+f)/2;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement d, f, m et d', f', m' les valeurs des variables d, f, m avant et après l'itération concernée.

* À moins de passer par le premier `if`, nous avons à chaque itération que $m = (d + f)/2$ et que $m' = (d' + f')/2$, ce qui correspond à une partie de notre invariant.

* Si $t_m > n$, $d' = d$ et $f' = m - 1$. Comme $d \leq f \leq \ell - 1$ et que $m = (d + f)/2$, nous avons bien que $d - 1 \leq f \leq \ell - 1$. Comme $0 \leq d \leq f + 1$, nous obtenons que $0 \leq d' \leq f + 1$. Or comme d' n'est pas modifié, nous pouvons également écrire que $d' - 1 \leq f' \leq \ell - 1$ et $0 \leq d' \leq f' + 1$.
* Si $t_m < n$, $d' = m + 1$ et $f' = f$. Comme $d \leq f \leq \ell - 1$ et que $m = (d + f)/2$, nous avons bien que $0 \leq d \leq f + 1$. Comme $d \leq f \leq \ell - 1$, nous obtenons que $d + 1 \leq f' \leq \ell - 1$. Or comme f' n'est pas modifié, nous pouvons également écrire que $d' - 1 \leq f' \leq \ell - 1$ et $0 \leq d' \leq f' + 1$.
* Si $t_m = n$, le premier `if` s'exécute et nous quittons la boucle. On garde bien que $0 \leq d \leq f - 1$, $d + 1 \leq f \leq \ell - 1$ ainsi que $m = (d + f)/2$.

– On peut sortir de la boucle de deux façons différentes. La première correspond à $\{I \wedge t_m = n\}$. Comme montré au point précédent, l'invariant tient et nous avons bien que la fonction retourne l'indice de n dans t.
Le deuxième cas est quand nous avons $\{I \wedge \neg \text{Cond}\}$. On a alors que $d > f$, ce qui signifie que notre sous-tableau actuel de t est le sous-tableau vide. n n'est donc pas présent dans t et -1 est retourné.

– Il reste à démontrer que l'exécution de la fonction se termine toujours. Il faut donc trouver un variant de boucle qui est un entier non négatif dont la valeur décroît strictement à chaque itération de la boucle :

$$v = f - d + 1$$

3. Voici un code correspondant à la fonction demandée :

```
void f(float *t, unsigned n){
    for(unsigned i = 1; i < n; i++){
        t[i] += t[i-1];
    }
}
```

- Complexité : Ce code parcourt une fois toutes les cases du tableau donné en entré de taille n. La complexité en temps sera donc de $\mathcal{O}(n)$.
- Invariant : Pour montrer que la fonction est correcte, on doit montrer que lorsque la fonction termine son exécution, son opération s'est déroulée correctement. On souhaite alors établir la validité du triplet :

$$\{t = t_0, \ldots, t_{n-1}, n > 0\}$$

```
for (unsigned i = 1; i < n; i++){
    t[i] += t[i-1];
}
```

$$\{t = t_0, t_0 + t_1, \ldots, \sum_{i=0}^{n-1} t_i\}$$

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

$$\{t = t_0, \ldots, t_{n-1}, n > 0, i = 1\}$$

```
while (i < n){
    t[i] += t[i-1];
    i++;
}
```

$$\{t = t_0, t_0 + t_1, \ldots, \sum_{i=0}^{n-1} t_i\}$$

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 :

$$\begin{aligned} I :& 0 \leq i \leq n \\ & \text{et } \forall j < i, t_j^* = \sum_{k=0}^{j} t_k \\ & \text{et } t = t^*0, \ldots, t_i^* - 1, t_i, \ldots, t_{n-1} \end{aligned}$$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, tous les éléments précédents l'élément d'indice i ont été modifiés en la somme des précédents éléments.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = 1$ et nous avons bien que tous les éléments précédents celui d'indice 1 (donc l'élément t_0) vaut $t_0^* = \sum_{j=0}^{0} t_j$.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, i < n\}$$

```
t[i] += t[i-1];
i++;
```

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement i et i' la valeur de la variable i avant et après l'itération concernée, ainsi que t et t' le tableau t avant et après l'itération concernée.
  - Dans tous les cas, nous avons que $i' = i + 1$. Comme $0 \leq i < n$, nous avons bien que $0 < i' \leq n$.
  - À chaque itération, nous laissons le tableau t inchangé à l'exception de l'élément d'indice i. Celui-ci devenant : $t_i^* = \sum_{j=0}^{i}$. Nous avons alors :

$$t = t_0^*, \ldots, t_{i-1}^*, t_i, \ldots t_{n-1}$$

$$t' = t_0^*, \ldots, t_{i-1}^*, t_i^*, t_{i+1}, \ldots t_{n-1}$$

    Par conséquent, on a :

$$t' = t_0^*, \ldots, t_{i'-1}^*, t_{i'}, \ldots t_{n-1}$$

- En fin de boucle, on a $\{I, \wedge \text{Cond}\}$, qui implique $i = n$. On a alors $\forall j < n, t_j^* = \sum_{k=0}^{j} t_j$ et comme $t = t_0^*, \ldots, t_{i-1}^*, t_i, \ldots, t_{n-1}$, on a bien que $t = t_0^*, \ldots, t_{n-1}^* = t_0, t_0 + t_1, \ldots, \sum_{i=0}^{n-1} t_i$.
- Il reste à démontrer que l'exécution de la fonction se termine toujours. Il faut donc trouver un variant de boucle qui est un entier non négatif dont la valeur décroît strictement à chaque itération de la boucle :

$$v = n - i$$

4. Voici un code correspondant à la fonction demandée :

```
double fact(unsigned n){
    double res = 1.0;
    for (unsigned i = 2; i <= n; i++){
        res *= i;
    }
    return res;
}
```

- Complexité : Ce code itère au maximum n fois. La complexité en temps sera donc de $\mathcal{O}(n)$.
- Invariant : Pour montrer que la fonction est correcte, on doit montrer que lorsque la fonction termine son exécution, la valeur renvoyée est correcte. On souhaite alors établir la validité du triplet :

$$\{n \geq 0, res = 1.0\}$$

```
for (unsigned i = 2; i <= n; i++)
    res *= i;
```

$$\{res = n! = \prod_{j=1}^{n} j\}$$

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

$$\{n \geq 0, res = 1.0, i = 2\}$$

```
while (i <= n){
    res *= i;
    i++;
}
```

$$\{res = n! = \prod_{j=1}^{n} j\}$$

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 :

$$I : 2 \leq i \leq n+1$$
$$\text{et } res = \prod_{j=1}^{i-1} j = (i-1)!$$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, la variable res contient la factorielle de $i-1$.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = 2$ et nous avons que $res = 1 = (i-1)!$.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, i <= n\}$$

```
res *= i;
i++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement i et i' la valeur de la variable i avant et après l'itération concernée, ainsi que res et res' la valeur de la variable res avant et après l'itération concernée.

* Nous avons que $i' = i + 1$. Comme $2 \leq i < n + 1$, nous avons bien que $2 \leq i' \leq n + 1$.
* À chaque itération, nous avons $res' = res * i$. Comme $res = (i-1)!$, nous avons $res' = (i-1)! * i = i!$. Par conséquent, on a : $res' = (i'-1)!$.

– En fin de boucle, on a $\{I, \neg \text{Cond}\}$, ce qui implique $i = n + 1$, nous avons bien alors que $res = (i-1)! = (n+1-1)! = n!$.

– Il reste à démontrer que l'exécution de la fonction se termine toujours. Il faut donc trouver un variant de boucle qui est un entier non négatif dont la valeur décroît strictement à chaque itération de la boucle :

$$v = n - i + 1$$

5. Voici un code correspondant à la fonction demandée :

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

- Complexité : Nous parcourons le tableau, élément par élément, 1 fois. Si le tableau est de taille n, alors la complexité théorique en temps est $\mathcal{O}(n)$.
- Invariant : Pour montrer que la fonction est correcte, on doit montrer que lorsque la fonction termine son exécution, son opération s'est déroulée correctement. On souhaite alors établir la validité du triplet :

$$\{t = t_0, \ldots, t_{n-1}, n > 0, *min = t_0, *max = t_0\}$$

```
for (unsigned i = 1; i < n; i++){
    if (t[i] < *min)
        *min = t[i];
    if (t[i] > *max)
        *max = t[i];
}
```

$$\{*min = \min(t_0, \ldots, t_{n-1}), *max = \max(t_0, \ldots, t_{n-1})\}$$

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

$$\{t = t_0,' ldots, t_{n-1}, n > 0, *min = t_0, *max = t_0, i = 1\}$$

```
while (i < n){
    if (t[i] < *min)
        *min = t[i];
    if (t[i] > *max)
        *max = t[i];
    i++;
}
```

$$\{*min = \min(t_0, \ldots, t_{n-1}), *max = \max(t_0, \ldots, t_{n-1})\}$$

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 :

$$\begin{aligned} I :& 1 \leq i \leq n \\ & \text{et } *min = \min(t_0, \ldots, t_{i-1}) \\ & \text{et } *max = \max(t_0, \ldots, t_{i-1}) \end{aligned}$$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, la variable $*min$ contient le plus petit élément entre t_0 et t_{i-1} alors que $*max$ contient le plus grand élément dans ce même sous-tableau.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = 1$, $*max = t_0$ et $*min = t_0$. On a bien que $*max$ et $*min$ contiennent respectivement le plus grand et le plus petit élément du sous-tableau désigné par le seul élément : t_0.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, i < n\}$$

```
if (t[i] < *min)
    *min = t[i];
if (t[i] > *max)
    *max = t[i];
i++
```

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement $i, *min, *max$ et $i', *min', *max'$ la valeurs des variables avant et après l'itération concernée.
  - Nous pour chaque itération que $i' = i + 1$. Comme $1 \leq i < n$, nous avons bien que $1 \leq i' \leq n$.
  - Si $t_i < *min$, alors $*min = t_i$. Comme $*min = \min(t_0, \ldots, t_{i-1})$, nous savons que t_i est le nouveau plus petit élément du sous-tableau : $*min' = \min(t_0, \ldots, t_{i-1}, t_i) = \min(t_0, \ldots, t_{i'-1})$.
  - Si $t_i > *max$, alors $*max = t_i$. Comme $*max = \max(t_0, \ldots, t_{i-1})$, nous savons que t_i est le nouveau plus grand élément du sous-tableau : $*max' = \max(t_0, \ldots, t_{i-1}, t_i) = \max(t_0, \ldots, t_{i'-1})$.
  - Dans les autres cas, $*min'$ et $*max'$ ne sont pas modifiés et contiennent toujours le plus petit et plus grand élément du sous tableau désigné par $t_0, \ldots, t_{i-1}, t_i$ qui équivaut au sous-tableau $t_0, \ldots, t_{i'-1}$.

– En fin de boucle, on a $\{I \wedge \neg \text{Cond}\}$, ce qui implique $i = n$, nous avons bien alors que $*min = \min(t_0, \ldots, t_{i-1}) = \min(t_0, \ldots, t_{n-1})$ et $*max = \max(t_0, \ldots, t_{i-1}) = \max(t_0, \ldots, t_{n-1})$
– Il reste à démontrer que l'exécution de la fonction se termine toujours. Il faut donc trouver un variant de boucle qui est un entier non négatif dont la valeur décroît strictement à chaque itération de la boucle :

$$v = n - i$$