Correction des exercices sur les invariants

1. Val: 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 etablir la validité du triplet :

$$\{\texttt{longueur} \geq 0, \texttt{pascal} = [p_{0,0}, p_{0,1}, \ldots, p_{0,\texttt{longueur}-1}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}],$$
$$p_{0,0} = 1, \forall i, j \in [1, \texttt{longueur} - 1] : p_{i,j} = 0\}$$

```
for (int i = 1; i < longueur; i++){
    pascal[i][0] = 1;
    pascal[i][i] = 1;
    for (int j = 1; j < i; j++)
        pascal[i][j] = pascal[i-1][j-1] +
                       pascal[i-1][j];
}
```

{`pascal` représente un triangle de pascal}

(a) Pour la boucle intérieure, en la décomposant, on obtient le triplet :

$$\{i \geq 1, j = 1, \texttt{pascal} = [p_{0,0}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}], p_{i_0} = 1, p_{i,i} = 1\}$$

```
while (j < i){
    pascal[i][j] = pascal[i-1][j-1] +
                   pascal[i-1][j];
    j++;
}
```

{La ligne i de `pascal` représente les éléments de la $i$ème ligne du triangle de Pascal}

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 :$i \geq 1$
et $1 \leq j \leq i$
et $\texttt{pascal} = [p_{0,0}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}]$
où $p_{i,1}, \ldots, p_{i,j-1}$ contiennent les éléments d'un triangle de Pascal de ligne i : $\forall k : 1 \leq k \leq j-1$ on a $p_{i,k} = p_{i-1,k-1} + p_{i-1,k}$

Cet invariant exprime qu'au début et à la fin de chaque itération, les éléments $p[0:i][0:j-1]$ correspondent aux éléments du triangle de Pascal.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i \geq 1$, ainsi que $j = 1$, et `pascal` $= [p_{0,0}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}]$. On a bien que $p_{i,0}$ est un élément du triangle de Pascal.
- Pour chaque itération de la boucle, on a le triplet :

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

```
pascal[i][j] = pascal[i-1][j-1] +
               pascal[i-1][j];
j++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement j et j', ainsi que p et p' la valeur des variables j et `pascal` avant et après l'itération concernée.

* On a $j' = j + 1$. Des contraintes $j \geq 1$ et $j < i$, on déduit $j' \geq 1$ et $j' \leq i$.
* On a `pascal` $= [p_{0,0}, \ldots, p'_{i,1}, \ldots, p'_{i,j-1}, \ldots p_{\texttt{longueur}-1,\texttt{longueur}-1}]$. Cela entraîne

$$\begin{aligned}\texttt{pascal'} &= [p_{0,0}, \ldots, p'_{i,1}, \ldots, p'_{i,j-1}, p'_{i,j}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}] \\ &= [p_{0,0}, \ldots, p'_{i,1}, \ldots, p'_{j'-1}, \ldots p_{\texttt{longueur}-1,\texttt{longueur}-1}]\end{aligned}$$

* L'itération va mettre dans l'élément de ligne i et de colonne j la somme entre l'élément de ligne $i-1$ et de colonne $j-1$ et l'élément de ligne $i-1$ et de colonne j. On a donc bien que $p_{i,j}$ a sa valeur dans le triangle de Pascal.
* En fin de boucle, on a $\{I, j \geq i\}$, qui implique $j = i$. On a bien alors `pascal` $= p_{0,0}, \ldots, p'_{i,1}, \ldots, p'_{i,j-1}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}$.

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 :

$$v = i - j$$

(b) En décomposant la boucle extérieure, on obtient le triplet :

$$\{\texttt{longueur} \geq 1, i = 1, \texttt{pascal} = [p_{0,0}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}]\}$$

```
while (i < longueur){
    pascal[i][0] = 1;
    pascal[i][i] = 1;
    // 2nd boucle
    i++;
}
```

{`pascal` $= [p_{0,0}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}]$ représente un triangle de Pascal}

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 :`longueur` ≥ 1

et $1 \leq i \leq$ `longueur`

et `pascal` $= [p_{0,0}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}]$,

où $p_{0,0}, \ldots, p_{i-1,i-1}$ contiennent les éléments du triangle de Pascal

Cet invariant exprime qu'au début et à la fin de chaque itération, les éléments $p[0:i-1][0:i-1]$ correspondent aux éléments du triangle de Pascal.

Montrons maintenant que cet invariant est valide.

- Initialement, on a `longueur` ≥ 1, ainsi que $i = 1$, et `pascal` $= [p_{0,0}, \ldots,]$. On a bien que $p[0][0 :$ `longueur` $- 1]$ sont des éléments du triangle de Pascal.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, i < \texttt{longueur}\}$$

```
pascal[i][0] = 1;
pascal[i][i] = 1;
// 2nd boucle
i++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement i et i', ainsi que p et p' la valeur des variables i et `tableau` avant et après l'itération concernée.

* On a $i' = i + 1$. Des contraintes $i \geq 1$ et $i <$ `longueur`, on déduit $i' \geq 1$ et $i' \leq$ `longueur`.

* On a `pascal` $= [p_{0,0}, \ldots, p'_{i-1,0}, \ldots, p'_{i-1,i-1}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}]$. Cela entraine

  $$\begin{aligned}\texttt{pascal'} &= [p_{0,0}, \ldots, p'_{i-1,0}, \ldots, p'_{i-1,i-1}, \ldots p_{\texttt{longueur}-1,\texttt{longueur}-1}] \\ &= [p_{0,0}, \ldots, p'_{i',0}, \ldots, p'_{i',i'}, \ldots p_{\texttt{longueur}-1,\texttt{longueur}-1}]\end{aligned}$$

* L'itération va mettre dans tous les éléments de ligne i leur valeur dans le triangle de Pascal. On a donc bien que $p_{0,0}, \ldots, p_{i,i}$ sont des éléments du triangle de Pascal.
* En fin de boucle, on a $\{I, i \geq \texttt{longueur}\}$, qui implique $i = \texttt{longueur}$. On a bien alors

  $$\texttt{pascal} = p_{0,0}, \ldots, p'_{i-1,0}, \ldots, p'_{i-1,i-1}, \ldots, p_{\texttt{longueur}-1,\texttt{longueur}-1}.$$

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 :

$$v = \texttt{longueur} - i$$

Compl: (a) remplir : Pour une valeur donnée de `longueur`, la fonction effectue `taille`-1 itérations qui exécutent chacune au plus `taille`-2 fois une partie de code en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(\texttt{longueur}^2)$.

(b) print : Pour une valeur donnée de `longueur`, la fonction effectue `taille` itérations qui exécutent chacune au plus `taille` fois une partie de code en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(\texttt{longueur}^2)$.

(c) main : Pour une valeur donnée de `longueur`, la fonction effectue 2 sous-fonctions de complexité $\mathcal{O}(\texttt{longueur}^2)$ chacune ainsi que `longueur` itérations qui exécutent chacune `longueur` fois une partie de code en temps constant. La complexité est donc $3 * \texttt{longueur}^2$, ce qui correspond à une complexité en temps $\mathcal{O}(\texttt{longueur}^2)$.

2. Val: 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 :

$$\{p1 = \texttt{caracteres}, p2 = \texttt{caracteres} + \ell\},$$
où ℓ correspond à la taille de `caracteres`

```
while (*p1){
    if (*p1 != *p2)
        return 0;
    p1++;
    p2--;
}
```

$\{$`caracteres` est un palindrome$\}$

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 implique la postcondition après la dernière itération. Un invariant possible est :

I :$p1 \geq$ `caracteres`
et $p2 \leq$ `caracteres` $+ \ell$, où ℓ est la taille de la chaîne de caractères
et $\forall p :$ `caracteres` $\leq p \leq p1 \implies *p \neq 0$
et $*(p1 - 1) = *(p2 + 1)$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, les éléments précédemment pointés par $p1$ et $p2$ sont identiques. L'invariant exprime aussi qu'il n'y a pas de caractères de terminaison dans les éléments parcourus par $p1$.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $p1 =$ `caracteres` et $p2 =$ `caracteres` $+ \ell$, et donc une chaîne de caractères vide est bien un palindrome.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, p1 \neq 0\}$$

```
if (*p1 != *p2)
    return 0;
p1++;
p2--;
```

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement $p1$ et $p1'$, ainsi que $p2$ et $p2'$ la valeur des variables $p1$ et $p2$ avant et après l'itéraiton concernée.
  - Dans tous les cas, on a $p1' = p1 + 1$ et $p2' = p2 - 1$, et la précondition implique que $*p1$ n'est pas nul.
  - Si $*p1 \neq *p2$, alors l'instruction située dans le `if` s'exécute, et la boucle se termine. On a bien que $*(p1-i-1) = *(p2+i-1)$, où i correspond au nombre d'itérations de la boucle.
  - Si $*p1 = *p2$, alors l'instruction située dans le `if` ne s'exécute pas, et on a bien que $*(p1' - i - 1) = *(p2' + i - 1)$, où i correspond au nombre d'itérations de la boucle.
- En fin de boucle, on a $\{I, p1 = 0\}$, ce qui implique bien que la chaîne de caractères commençant par l'endroit pointé par `caracteres` et se terminant au premier caractère terminateur forme un palindrome.

Compl: Pour une chaîne de caractères de taille ℓ, la fonction effectue au plus ℓ itérations qui s'exécutent chacune en temps constant. La complexité en temps de la donction est donc $\mathcal{O}(\ell)$.

3. Val: 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 :

$\{\texttt{longueur} \geq 0, \texttt{tableau} = [t_0, t_1, \ldots, t_{n-1}]\}$

```
for (int i = 0; i < longueur; i++){
    for(int j = 0; j < i; j++){
        if (tableau[i] < tableau[j]){
            int tmp = tableau[j];
            tableau[j] = tableau[i];
            tableau[i] = tmp;
            }
    }
}
```

$\{\texttt{tableau} = [t_0, t_1, \ldots, t_{\texttt{longueur}-1}],\ \text{où } t_0 \leq t_1 \leq \ldots \leq t_{\texttt{longueur}-1}\}$

(a) Pour la boucle intérieure, en la décomposant, on obtient le triplet :

$\{i \geq 0, j = 0, \texttt{tableau} = [t_0, t_1, \ldots, t_{i-1}, t_i, \ldots, t_{\texttt{longueur}-1}\}$

```
while(j < i){
    if (tableau[i] < tableau[j]){
        int tmp = tableau[j];
        tableau[j] = tableau[i];
        tableau[i] = tmp;
    }
    j++;
}
```

$\{\text{tableau} = [t_0, t_1, \ldots, t_{i-1}, t_i, \ldots t_{\texttt{longueur}-1}],\ \text{où } t_0 \leq t_1 \leq \ldots \leq t_{i-1}\}$

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 :

I :$i \geq 0$
et $0 \leq j \leq i$
et $\texttt{tableau} = [t_0, t_1, \ldots, t_{j-1}, \ldots t_{i-1}, \ldots, t_{\texttt{longueur}-1}]$,
où $t_0 \leq t_1 \leq \ldots \leq t_{j-1}$

Cet invariant exprime qu'au début et à la fin de chaque itération, les éléments `tableau[0 : j − 1]` sont triés.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i \geq 0$, ainsi que $j = 0$, et `tableau`$= [t_0, t_1, \ldots, t_{i-1}, \ldots, t_{n-1}]$. On a bien qu'un tableau vide est trié.
- Pour chaque itération de la boucle, on a le triplet :

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

```
if (tableau[i] < tableau[j]){
      int tmp = tableau[j];
      tableau[j] = tableau[i];
      tableau[i] = tmp;
}
j++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement t et t', ainsi que j et j' la valeur des variables `tableau` et j avant et après l'itération concernée.

* On a $j' = j + 1$. Des contraintes $j \geq 0$ et $j < i$, on déduit $j' \geq 0$ et $j' \leq i$.
* On a `tableau` $= [t'_0, t'_1, \ldots, t'_{j-1}, t_j, \ldots, t_{i-1}, \ldots, t_{n-1}]$, où $t'_0 \leq t'_1 \leq \ldots \leq t'_{j-1}$. Cela entraine

$$\begin{aligned} \texttt{tableau'} &= [t'_0, t'_1, \ldots, t'_{j-1}, t'_j, \ldots, t_{i-1}, \ldots, t_{n-1} \\ &= [t'_0, t'_1, \ldots, t'_{j'-1}, \ldots, t_{i-1}, \ldots, t_{n-1} \end{aligned}$$

* Si $t_i < t_j$, l'itération laisse les éléments du `tableau` inchangés, à l'exception de celui d'indice i et j qui sont échangés.
* Si $t_i \geq t_j$, l'itération laisse les éléments du `tableau` inchangés. Ces 2 points vérifient la postcondition.
* En fin de boucle, on a $\{I, j \geq i\}$, qui implique $j = i$. On a bien alors `tableau` $= [t'_0, t'_1, \ldots, t'_{i-1}, \ldots, t_{n-1}]$.

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 :

$$v = i - j$$

(b) En décomposant la boucle intérieure, on obtient le triplet :

$$\{\texttt{longueur} \geq 0, i = 0, \texttt{tableau} = [t_0, t_1, \ldots, t_{i-1}, t_i, \ldots, t_{\texttt{longueur}-1}\}$$

```
while (i < longueur){
    //2nd boucle
    i++;
}
```

$\{\text{tableau} = [t_0, t_1, \ldots t_{\texttt{longueur}-1}],\ \text{où } t_0 \leq t_1 \leq \ldots \leq t_{n-1}\}$

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 :

I :`longueur` ≥ 0
et $0 \leq i \leq$ `longueur`
et `tableau` $= [t_0, t_1, \ldots, t_{i-1}, \ldots, t_{\texttt{longueur}-1}]$,
où $t_0 \leq t_1 \leq \ldots \leq t_{i-1}$

Cet invariant exprime qu'au début et à la fin de chaque itération, les éléments `tableau`$[0 : i-1]$ sont triés.

Montrons maintenant que cet invariant est valide.

- Initialement, on a `longueur` ≥ 0, ainsi que $i = 0$, et `tableau`$= [t_0, t_1, \ldots, t_{n-1}]$. On a bien qu'un tableau vide est trié.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, i < \texttt{longueur}\}$$

$$\texttt{i++;}$$

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement t et t', ainsi que i et i' la valeur des variables `tableau` et i avant et après l'itération concernée.
  - On a $i' = i + 1$. Des contraintes $i \geq 0$ et $i <$ `longueur`, on déduit $i' \geq 0$ et $i' \leq$ `longueur`.
  - On a `tableau` $= [t'_0, t'_1, \ldots, t'_{i-1}, \ldots, t_{n-1}]$, où $t'_0 \leq t'_1 \leq \ldots \leq t'_{i-1}$. Cela entraine

$$\begin{aligned}\texttt{tableau'} &= [t'_0, t'_1, \ldots, t'_{i-1}, t'_i, \ldots, t_{n-1}] \\ &= [t'_0, t'_1, \ldots, t'_{i'-1}, \ldots, t_{n-1}]\end{aligned}$$

  - L'itération va trier les éléments entre `tableau`$[0]$ et `tableau`$[i]$ et laisser les autres éléments inchangés. On a donc bien $t_0 \leq t_1 \leq \ldots \leq t_i$.

* En fin de boucle, on a $\{I, i \geq \texttt{longueur}\}$, qui implique $i = \texttt{longueur}$. On a bien alors $\texttt{tableau} = [t'_0, t'_1, \ldots, t'_{n-1}]$.

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 :

$$v = \texttt{longueur} - i$$

Compl: Pour une valeur `longueur` donnée, la fonction effectue `longueur` itération qui exécutent chacune au maximum $\texttt{longueur} - 1$ fois une partie de code en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(\texttt{longueur}^2)$.

4. Val: 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 :

$$\{n \geq 0, \texttt{sum} = 0, \texttt{t} = [t_0, t_1, \ldots, t_{n-1}]\}$$

```
for (int i = 0; i < n; i++){
    sum += t[i];
    t[i] = sum;
}
```

$$\{\texttt{t} = [t_0, t_0 + t_1, t_0 + t_1 + t_2, \ldots, t_0 + t_1 + \ldots + t_{n-1}]\}$$

En décomposant la boucle `for`, on obtient le triplet équivalent :

$$\{n \geq 0, i = 0, \texttt{sum} = 0, \texttt{t} = [t_0, t_1, \ldots, t_{n-1}]\}$$

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

$$\{\texttt{t} = [t_0, t_0 + t_1, t_0 + t_1 + t_2, \ldots, t_0 + t_1 + \ldots + t_{n-1}]\}$$

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 :

$$
\begin{aligned}
I :& n \geq 0 \\
& \text{et } 0 \leq i \leq n \\
& \text{et } \texttt{sum} = \sum_{j=0}^{i-1} t_j \\
& \text{et } \texttt{t}[t_0, t_1, \ldots, t_{n-1}] \\
& \text{et } \forall j : 0 \leq j \leq i-1 \text{ on a } t_j = \sum_{k=0}^{j} t_k
\end{aligned}
$$

Cet invariant exprime qu'au début et à la fin de chaque itération, `sum` contient la somme de tous les éléments déjà visités. En outre, les i premiers éléments du tableau `t` contiennent les sommes partielles déjà calculées, et les éléments suivants ont gardé leur valeur initiale.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $n \geq 0$, ainsi que $i = 0, \texttt{sum} = 0$ et $\texttt{t} = [t_0, t_1, t_2, \ldots, t_{n-1}]$. On a bien que la somme de 0 éléments est 0 et qu'il n'y a pas d'élément avant t_0.
- Pour chaque itération de la boucle, on a le triplet :

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

```
sum += t[i];
t[i] = a;
i++;
```

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement `sum` et `sum'`, ainsi que i et i', les valeur des variables `sum` et i avant et après l'itération concernée.

  * On a $i' = i + 1$. Des contraintes $i \geq 0$ et $i < n$, on déduit $i' \geq 0$ et $i' \leq n$.
  * On a $\texttt{sum'} = \texttt{sum} + \gamma$, où γ est la valeur de l'élément d'indice i du tableau t. Cela entraine

$$
\begin{aligned}
\texttt{sum'} &= t_0 + t_1 + t_2 + \ldots + t_i = \sum_{j=0}^{i} t_j \\
&= t_0 + t_1 + t_2 + \ldots + t_{i'-1} = \sum_{j=0}^{i'-1} t_j
\end{aligned}
$$

* L'itération laisse les éléments du tableau `t` inchangés, à l'exception de celui d'indice i qui est remplacé par la valeur de `sum'`. Par conséquent, on a

$$\begin{aligned}\texttt{t'} &= [t_0, t_0 + t_1, t_0 + t_1 + \ldots + t_{i'-1}, t_{i+1}, t_{i+2}, \ldots, t_{n-1}] \\ &= [t_0, t_0 + t_1, t_0 + t_1 + \ldots + t_{i'-1}, t_{i'}, t_{i'+1}, \ldots, t_{n-1}]\end{aligned}$$

  La postcondition est donc bien vérifiée.
* En fin de boucle, on a $\{I, i \geq n\}$, qui implique $i = n$. On a bien alors

$$\texttt{t} = [t_0, t_0 + t_1, t_0 + t_1 + t_2, \ldots, t_0 + t_1 + \ldots + t_{n-1}]$$

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 :

$$v = n - i$$

Compl: Pour une valeur donnée de n, la fonction effectue n itérations qui s'exécutent chacune en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(n)$.

5. Val: 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 :

$$\{\texttt{taille} \geq 0\}$$

```
for (n = 0; n < taille && t1[n] == t2[n]; n++);
return n;
```

$\{n =$ longueur du plus grand préfixe commun de `*t1` et `*t2`$\}$

En décomposant la boucle `for`, on obtient le triplet équivalent :

$$\{\texttt{taille} \geq 0, n = 0\}$$

```
while (n < taille && t1[n] == t2[n])
    n++;
```

$\{n =$ longueur du plus grand préfixe commun de `*t1` et `*t2`$\}$

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 :

$$\begin{aligned} I :& \texttt{taille} \geq 0 \\ & \text{et } n \leq \texttt{taille} \\ & \text{et } \texttt{t1}[0 : n-1] = \texttt{t2}[0 : n-i], \end{aligned}$$

où `t[`$x : y$`]` dénote un tableau contenant les éléments de `t` compris entre les indices x et y inclus.

Montrons maintenant que cet invariant est valide.

- Initialement, on a `taille` ≥ 0, et $n = 0$. Et on a bien que 2 tableaux vides sont égaux.
- Pour chaque itération de la boucle, on a le triplet :

$$\{I, n < taille, \texttt{t1}[n] == \texttt{t2}[n]\}$$

$$\text{i++;}$$

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement n et n' la valeur de la variable n avant et après l'itération. On a :

$$n' = n + 1$$

  donc

$$\texttt{t1}[0 : n' - 1] = \texttt{t2}[0 : n' - 1]$$

- En fin de boucle, on a $\{I, i \geq \texttt{taille}$ ou $\texttt{t1}[n] \neq \texttt{t2}[n]\}$, ce qui implique bien que la valeur de n est égale àa la longueur du plus grand préfixe commun de `*t1` et `*t2`. En effet,
  - Si I et $i \geq \texttt{taille}$ est vrai, alors on a

$$n = \texttt{taille} \text{ et } \texttt{t1}[0 : \texttt{taille} - 1] = \texttt{t2}[0 : \texttt{taille} - 1]$$

    et la longueur du plus grand préfixe commun de `*t1` et `t2` est bien de longueur $n =$ `taille`
  - Si I et $\texttt{t1}[n] \neq \texttt{t2}[n]$ est vrai, alors on a

$$\texttt{t1}[0 : n-1] = \texttt{t2}[0 : n-1] \text{ et } \texttt{t1}[n] \neq \texttt{t2}[n]$$

  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 :

$$v = \texttt{taille} - n$$

Compl: Pour une valeur donnée de `taille`, la fonction effectue au plus `taille` itérations qui s'exécutent chacune en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(\texttt{taille})$.

6. Val: 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 :

$$\{c = c_0\}$$

```
for (total = 0; *c; c++)
    if (*c >= '0' && *c <= '9')
        total += *c - '0';
```

$\{\texttt{total} =$ total des chiffres contenus dans la chaine $c_0\}$

En décomposant la boucle `for`, on obtient le triplet équivalent :

$$\{c = c_0, \texttt{total} = 0\}$$

```
while (*c){
    if (*c >= '0' && *c <= '9')
        total += *c - '0';
    c++;
}
```

$\{\texttt{total} =$ total des chiffres contenus dans la chaine $c_0\}$

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 :$c \geq c_0$ et `total` = total des chiffres contenus dans la chaîne $[*c_0 * (c_0 + 1) * (c_0 + 2) \ldots * (c - 1)]$
et $\forall d : c_0 \leq d \leq c \implies *d \neq 0$

En d'autres termes, cet invariant exprime qu'avant et après chaque itération de la boucle, la valeur de `total` correspond au total des chiffres appartenant à un préfixe de la chaîne fournie à la fonction. Ce préfixe commence à l'endroit désigné par le pointeur c_0 passé en argument à la fonction, et se termine immédiatement avec le caractère pointé par la valeur courante de c. L'invariant exprime aussi que ce préfixe ne contient aucun caractère de terminaison.

Montrons maintenant que cet invariant est valide.

– Initialement, on a $c = c_0$, et donc le préfixe considéré est vide. On a alors bien `total` $= 0$.
– Pour chaque itération de la boucle, on a le triplet :

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

```
    if (*c >= '0' && *c <= '9')
        total += *c - '0';
    c++;
}
```

$$\{I\}$$

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

* Dans tous les cas, on a $c' = c + 1$, et la chaîne de caractères est $*c_0 * (c_0 + 1) * (c_0 + 2) \ldots * c$. La précondition implique que ce caractère $*c$ n'est pas nul.
* Si $*c$ est un chiffre, alors l'instruction située dans le `if` s'exécute, et on a `total'` $=$ `total` $+ \gamma$, où γ est la valeur numérique du chiffre en question. La postcondition est satisfaire dans ce cas.
* Si $*c$ n'est pas un chiffre, alors l'instruction située dans le `if` ne s'exécute pas, et on a `total'` $=$ `total`. La postcondition est aussi satisfaite dans ce cas.

– En fin de boucle, on a $\{I, *c = 0\}$, ce qui implique que la valeur de `total` est égale au total des chiffres contenus dans la chaîne de caractères commençant à l'endroit pointé par c_0, et se terminant au premier caractère terminateur.

Compl: Pour une chaine de caractères de taille ℓ, la fonction effectue au plus ℓ itérations qui s'exécutent chacune en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(\ell)$.