# Correction des exercices du chapitre 3

1. (a) Montrons que ce triplet est valide ou non, en notant respectivement $x$ et $x'$, la valeur de la variable $x$ avant et après l'instruction concernée.
   - On a $x' = x - 1$.
   - Si $x > 0$, alors après avoir effectué l'instruction, on obtient bien que $x' \geq 0$.

   Le triplet est donc valide.

   (b) Montrons que ce triplet est valide ou non, en notant respectivement $x$ et $x'$, la valeur de la variable $x$ avant et après l'instruction concernée.
   - On a $x' = x + 1$.
   - Si $x < 2^{31} - 1$, alors après avoir effectué l'instruction, on obtient bien que $x' > 0$.
   - Si $x = 2^{31} - 1$, alors après avoir effectué l'instruction, on obtient que $x' = -2^{31}$.

   Le triplet n'est donc pas valide.

   (c) Montrons que ce triplet est valide ou non. Comme la précondition est toujours satisfaite, l'instruction s'exécutera toujours, peu importe la valeur de $x$. Notons respectivement $x$ et $x'$, la valeur de la variable $x$ avant et après l'instruction concernée.
   - Si $0 \leq x < 2^{31} - 1$, alors après avoir effectué l'instruction, on obtient bien que $x' > 0$.
   - Si $x < 0$, alors après avoir effectué l'instruction, on obtient que $x' \leq 0$.
   - Si $x = 2^{31} - 1$, alors après avoir effectué l'instruction, on obtient que $x' = -2^{31}$.

   Le triplet n'est donc pas valide.

   (d) Montrons que ce triplet est valide ou non. Comme la précondition ne peut jamais être satisfaite, il n'existe pas d'exécution pour laquelle la précondition est initialement vraie, l'instruction ne sera donc jamais éxécutée.
   Le triplet est donc valide.

   (e) Montrons que ce triplet est valide ou non, en notant respectivement $x$ et $x'$, la valeur de la variable $x$ avant et après l'instruction concernée.
   - On a $x' = x/2$ et $x > 0$.
   - Si $x\%2 = 0$, une nouvelle itération de la boucle est exécutée.
   - Si $x\%2 \neq 0$, on sort de la boucle et on obtient donc bien que $x$ est impair.

   Le triplet est donc valide.

2. (a) Nous avons l'invariant :

$$I : 2 \leq i \leq n+1$$
$$\text{et } \texttt{fact} = \prod_{2 \leq j < i} j$$

Montrons que cet invariant est valide. Pour ça, définissons d'abord la précondition et la postcondition :

$$P : \{i = 2, \text{fact} = 1, n > 1\}$$
$$Q : \{\text{fact} = \prod_{1 \leq j \leq n} j\}$$

- Initialement, on a $i = 2$ et `fact` $= 1$. On a donc bien que $\prod_{2 \leq j < 2} j = 1$. La précondition implique l'invariant.
- Pour chaque itération de la boucle, on a le triplet :

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

```
fact *= i;
i++;
```

$$\{I\}$$

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

  - On a $i' = i + 1$. Des contraintes $i \geq 2$ et $i \leq n$, on déduit $2 \leq i' \leq n + 1$
  - On a `fact'` $=$ `fact` $* i$. Cela entraine :

$$\texttt{fact'} = \prod_{2 \leq j < i+1} j$$
$$= \prod_{2 \leq j < i'} j$$

- En fin de boucle, on a $\{I, i > n\}$, ce qui implique que la valeur de $i = n + 1$ et celle de `fact` $= \prod_{2 \leq j < n+1} j$, qui est la même chose que $\prod_{1 \leq j \leq n} j$. Nous avons bien que l'invariant et la négation de la condition impliquent la postcondition.

L'invariant est donc bien valide.

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

$$v = n - i + 1$$

3. (1.2) (a) Pour montrer que le programme est correct, on doit montrer que lorsque le programme termine son exécution, les valeurs des variables sont correctes. On souhaite alors établir la validité du triplet suivant :

$$\{n \geq 0, k > 0, \texttt{multifactorielle = 1}\}$$

```
for (int i = n ; i > 0 ; i -= k )
    multifactorielle *= i ;
```

$$\{\texttt{multifactorielle} = \prod_{0 \leq j \leq \lfloor n/k \rfloor} (n - jk)\}$$

En décomposant la boucle `for`, on obtient :

$$\{n \geq 0, k > 0, i = n, \texttt{multifactorielle} = 1\}$$

```
while (i > 0){
    multifactorielle *= i;
    i -= k;
}
```

$$\{\texttt{multifactorielle} = \prod_{0 \leq j \leq \lfloor n/k \rfloor} (n - jk)\}$$

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 :

$$\begin{aligned} I : & n \geq 0 \\ & \text{et } -k + 1 \leq i \leq n \\ & \text{et } \forall j \in [0, n/k] : i = n - jk \\ & \text{et } \texttt{multifactorielle} = \prod_{0 \leq j < (n-i)/k} (n - jk) \end{aligned}$$

Cet invariant exprime qu'au début et à la fin de chaque itération, `multifactorielle` contient le produit des $n - jk$, pour tout $j$ entre 0 et $(n - i)/k$.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = n$, $k > 0$, ainsi que `multifactorielle` $= 1$. On a donc bien que $\prod_{0 \leq j < 0}(n - jk) = 1$
- Pour chaque itération de boucle, on a le triplet :

$$\{I, i > 0\}$$

```
multifactorielle *= i;
i -= k;
```

$\{I\}$

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

- On a $i' = i - k$. Des contraintes $i = n$, $i > 0$, on déduit que $i' \leq n$ et $i' > -k$.
- On a $\texttt{multifactorielle'} = \texttt{multifactorielle} * i$. Cela entraine :

$$\begin{aligned}\texttt{multifactorielle'} &= \texttt{multifactorielle} * i \\ &= \texttt{multifactorielle} * (i' - k)\end{aligned}$$

- L'itération multiplie la valeur actuelle de `multifactorielle` par $i$. Comme $i = (n - jk)$ pour tout $0 \leq j \leq n/k$, on a $\texttt{multifactorielle'} = \prod i$, ce qui vérifie la postcondition.

- En fin de boucle, on a $\{I, i \leq 0\}$, qui implique $-k+1 \leq i \leq 0$. On a bien alors $\prod_{0 \leq j \leq n/k}(n - jk)$.

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

$$v = \lfloor \frac{i}{k} \rfloor + 1$$

(b) Pour une valeur $n$ et $k$ donnée, la fonction effectue $n/k$ itérations en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(n)$.

(1.3) (a) Pour montrer que le programme est correct, on doit montrer que lorsque le programme termine son exécution, les valeurs des variables sont correctes. On souhaite alors établir la validité du triplet suivant :

$\{\#\texttt{nb} > 0, \texttt{petit} = \texttt{nb}, \texttt{grand} = \texttt{nb}, \texttt{somme} = \texttt{nb}, \texttt{produit} = \texttt{nb}, \texttt{nb\_nb} = 1\}$

```
while (scanf("%f", &nb){
    if (nb < petit)
        petit = nb;
    if (nb > grand)
        grand = nb;
    somme += nb;
    produit *= nb;
    nb_nb++;
}
```

$$\{\texttt{somme} = \sum_{0 \leq i < \#\texttt{nb}} \texttt{nb}_i, \texttt{produit} = \prod_{0 \leq i < \#\texttt{nb}} \texttt{nb}_i, \texttt{petit} = \texttt{min}(\texttt{nb}_{\#\texttt{nb}-1}, \ldots, \texttt{nb}_0),$$
$$\texttt{grand} = \texttt{max}(\texttt{\#nb-1}, \ldots, \texttt{nb}_0), \texttt{nb\_nb} = \#\texttt{nb}\}$$

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 :

$$\begin{aligned} I :& \#\texttt{nb} > 0 \\ & \text{et } \forall i \in [0, \#\texttt{nb}[\texttt{somme} = \sum_{0 \leq j < i} j \\ & \text{et } \forall i \in [0, \#\texttt{nb}[\texttt{produit} = \prod_{0 \leq j < i} j \\ & \text{et } 1 \leq \texttt{nb\_nb} \leq \#\texttt{nb} \\ & \text{et } \forall i \in [0, \#\texttt{nb}[\texttt{petit} = \texttt{min}(\texttt{nb}_i, \ldots, \texttt{nb}_0) \\ & \text{et } \forall i \in [0, \#\texttt{nb}[\texttt{grand} = \texttt{max}(\texttt{nb}_i, \ldots, \texttt{nb}_0) \end{aligned}$$

Cet invariant exprime qu'au début et à la fin de chaque itération, `somme` contient la somme de tous les nombres donnés en entrée, `produit` contient le produit de tous les nombres donnés en entrée, `petit` contient le plus petit réel des nombres donnés en entrée, `grand` contient le plus grand réel des nombres donnés en entrée, `nb_nb` contient le nombre de réels entrés.

Montrons maintenant que cet invariant est valide.

- Initialement, on a `petit` = `nb`, `grand` = `nb`, `somme` = `nb`, `produit` = `nb`, ainsi que `nb_nb` = 1. On a bien que le nombre actuel d'éléments est 1 et du coup, `petit`, `grand`, `somme`, et `produit` correspondent à cet élément.
- Pour chaque itération de boucle, on a le triplet :

$$\{I, \texttt{scanf}\}$$

```
if (nb < petit)
    petit = nb;
if (nb > grand)
    grand = nb;
somme += nb;
produit *= nb;
nb_nb++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en prenant en compte la valeur des variables avant et après l'itération concernée.

- Dans tous les cas, on a $\texttt{somme'} = \texttt{somme} + \texttt{nb}_i = \sum_{0 \leq j \leq i} nb_j = \sum_{0 \leq j < i'} nb_j$ $\texttt{produit'} = \texttt{produit} + \texttt{nb}_i = \prod_{0 \leq j \leq i} j = \prod_{0 \leq j < i'} j$, ainsi que $\texttt{nb\_nb'} = \texttt{nb\_nb} + 1$.
- Si $\texttt{nb}_i < \texttt{petit}$, $\texttt{petit'} = \texttt{nb}_i$.
- Si $\texttt{nb}_i > \texttt{grand}$, $\texttt{grand'} = \texttt{nb}_i$.

Ce qui vérifie la postcondition.

- En fin de boucle, on a $\{I, \neg\texttt{scanf}\}$ qui implique que plus aucun nombre n'est donné en entrée. On a bien alors

$$\{\texttt{somme} = \sum_{0 \leq i < \#\texttt{nb}} \texttt{nb}_i, \texttt{produit} = \prod_{0 \leq i < \#\texttt{nb}} \texttt{nb}_i, \texttt{petit} = \texttt{min}(\texttt{nb}_{\#\texttt{nb}-1}, \ldots, \texttt{nb}_0), \texttt{grand} = \texttt{max}(\texttt{\#nb-1}, \ldots, \texttt{nb}_0), \texttt{nb\_nb} = \#\texttt{nb}\}$$

(b) Pour une valeur $\#\texttt{nb}$ donnée, la fonction effectue $\#\texttt{nb}-1$ itérations en temps constant. La complexité en temps de la fonction est donc $\mathcal{O}(\#\texttt{nb})$.

(1.5) (a) Pour montrer que le programme est correct, on doit montrer que lorsque le programme termine son exécution, les valeurs des variables sont correctes. On souhaite alors établir laa validité du triplet suivant :

$$\{\texttt{nb} \geq 0, \texttt{log} = 0\}$$

```
for (int puissance2 = 2; puissance2 <= nb; log++){
    puissance2 *= 2;
}
```

$$\{\texttt{log} = \lfloor \texttt{log}_2 n \rfloor\}$$

En décompossant la boucle `for`, on obtient :

$$\{\textit{puissance}2 = 2, \texttt{nb} \geq 0, \texttt{log} = 0\}$$

```
while (puissance2 <= nb){
    puissance2 *= 2;
    log++;
}
```

$$\{\texttt{log} = \lfloor \texttt{log}_2 n \rfloor\}$$

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 :

$$\begin{aligned} I :& 2 \leq \texttt{puissance2} \leq 2n \\ & \text{et } n \geq 0 \\ & \text{et } \texttt{log} = \log_2(\texttt{puissance2}/2) \end{aligned}$$

Montrons maintenant que cet invariant est valide.

- Initialement, on a $\texttt{log} = 0$, ainsi que $\texttt{puissance2} = 2$. On a bien que $\log_2(1) = 0$ et que $2 \leq \texttt{puissance2} \leq 2n$.
- Pour chaque itération de boucle, on a le triplet :

$$\{I, \texttt{puissance2} \leq n\}$$

```
puissance *= 2;
log++;
```

$$\{I\}$$

Montrons que ce triplet est valide, en notant respectivement `log` et `log'`, ainsi que `puissance2` et `puissance2'` la valeur des variables `log` et `puissance2` avant et après l'itération concernée.

- On a $\texttt{puissance2'} = \texttt{puissance2} * 2$. Des contraintes $\texttt{puissance2} = 2$ et $\texttt{puissance2} \leq n$, on déduit $2 \leq \texttt{puissance2} \leq 2n$.
- On a $\texttt{log'} = \texttt{log} + 1$. Comme $\texttt{log} = \log_2(\texttt{puissance2}/2)$, Cela entraîne :

$$\begin{aligned} \texttt{log'} &= \log_2(\texttt{puissance2}/2) + 1 \\ &= \log_2(\texttt{puissance2}/2) + \log_2 2 \\ &= \log_2(2 * \texttt{puissance2}/2) \\ &= \log_2(\texttt{puissance2'}/2) \end{aligned}$$

Ce qui valide la postcondition.

- En fin de boucle, on a $\{I, \texttt{puissance2} > n\}$, qui implique $n < \texttt{puissance2} \leq 2n$. On a bien alors $\texttt{log} = \lfloor \log_2 n \rfloor = \log_2(\texttt{puissance2}/2)$.

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

$$I = 2n - \texttt{puissance2}$$

(b) Pour une valeur `nb` donnée, le programme effectue $\log_2(\texttt{nb})$ itérations en temps constant. La complexité en temps du programme est donc $\mathcal{O}(\texttt{log(nb)})$.

(1.6) (a) Pour montrer que le programme est correct, on doit montrer que lorsque le programme termine son exécution, les valeurs des variables sont correctes. On souhaite alors établir la validité du triplet suivant :

$$\{\texttt{exposant} \geq 0, \texttt{modulo} > 0, \texttt{resultat} = 1\}$$

```
for (int i = 0; i < exposant; i++){
    resultat = (resultat * base) % modulo;
}
```

$$\{\texttt{resultat} = \texttt{base}^{\texttt{exposant}} \bmod \texttt{modulo}\}$$

En décomposant la boucle `for`, on obtient :

$$\{\texttt{exposant} \geq 0, \texttt{modulo} > 0, \texttt{resultat} = 1, i = 0\}$$

```
while (i < exposant){
    resultat = (resultat * base) % modulo;
    i++;
}
```

$$\{\texttt{resultat} = \texttt{base}^{\texttt{exposant}} \bmod \texttt{modulo}\}$$

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 :

$$\begin{aligned} I : & \texttt{exposant} \geq 0 \\ & \text{et } 0 \leq i \leq \mathit{exposant} \\ & \text{resultat} = \prod_{0 \leq j < i} \texttt{base} \bmod \texttt{modulo} \end{aligned}$$

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = 0$, ainsi que $\texttt{resultat} = 1$. On a bien que resultat $= \prod_{0 \leq j < 0} \texttt{base} \bmod \texttt{modulo} = 1$
- Pour chaque itération de boucle, on a le triplet :

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

```
resultat = (resultat * base) % modulo;
i++;
```

$\{I\}$

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

- On a $i' = i + 1$. Des contraintes $i = 0$ et $\texttt{exposant} \geq 0$, on déduit $0 \leq i \leq \texttt{exposant}$
- On a $\texttt{resultat'} = \texttt{resultat} * \texttt{base} \bmod \texttt{modulo} = \prod_{0 \leq j < j+1} \texttt{base} \bmod \texttt{modulo} = \prod_{0 \leq j < i'} \texttt{base} \bmod \texttt{modulo}$, ce qui vérifie la postcondition.

• En fin de boucle, on a $\{I, i \geq \texttt{exposant}\}$, ce qui implique $i = \texttt{exposant}$. On a alors bien que $\texttt{resultat} = \prod_{0 \leq j < \texttt{exposant}} \texttt{base} \bmod \texttt{modulo} = \texttt{base}^{\texttt{exposant}} \bmod \texttt{modulo}$.

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

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

(b) Pour une valeur `exposant` donnée, le programme effectue `exposant` itérations en temps constant. La complexité en temps du programme est donc $\mathcal{O}(\texttt{exposant})$.

(2.5) (a) Pour montrer que le programme est correct, on doit montrer que lorsque le programme termine son exécution, les valeurs des variables sont correctes. On souhaite alors établir la validité du triplet suivant :

$$\{n \geq 2, f = 1, f\_1 = 1\}$$

```
for (int i = 2; i < n; i++){
    int tmp = f;
    f += f_1;
    f_1 = tmp;
}
```

$$\{f_n = f_{n-1} + f_{n-2}\}$$

En décomposant la boucle `for`, on obtient :

$$\{n \geq 2, f = 1, f\_1 = 1, i = 2\}$$

```
while (i < n){
    int tmp = f;
    f += f_1;
    f_1 = tmp;
    i++;
}
```

$$\{f_n = f_{n-1} + f_{n-2}\}$$

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 :

$$\begin{aligned} I : & n \geq 2 \\ & \text{et } 2 \leq i \leq n \\ & \text{et } f\_1_i = f_{i-1} \\ & \text{et } f\_1_i = f_{i-1} \end{aligned}$$

Cet invariant exprime qu'au début et à la fin de chaque itération, $f$ contient le nombre de Fibonacci $i$, $f\_1$ le nombre de Fibonacci $i-1$.

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = 2$, $n \geq 0$, ainsi que $f\_1 = 1$ et $f = 1$. On a alors bien que $f = 1 = \text{Fibonacci}_2$ et que $f\_1 = \text{Fibonacci}_1$.
- Pour chaque itération de boucle, on a le triplet :

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

  ```
  int tmp = f;
  f += f_1;
  f_1 = tmp;
  i++;
  ```

  $$\{I\}$$

  Montrons que ce triplet est valide, en notant la variable et la variable', la valeur des variables avant et après l'itération concernée.
  - On a $i' = i + 1$. Des contraintes $i = 2$ et $i < n$, on déduit $2 \leq i \leq n$.
  - On a $f' = f + f\_1$
  - On a $f\_1' = f$
  - L'itération calcule le nombre de Fibonacci dans $f'$ puis prépare les valeurs pour le prochain nombre de Fibonacci dans $f\_1$. Par conséquent, on a bien que $f_{i'} = f_i + f\_1_i$ et que $f\_1_{i'} = f_{i'-1} = f_i$. La postcondition est bien vérifiée.
- En fin de boucle, on a $\{I, i \geq n\}$, qui implique $i = n$. On a bien alors $f_n = f_{n-1} + f\_1_{n-1} = f_{n-1} + f_{n-2} = \text{Fibonacci}_n$.

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

$$v = n - i$$

(b) Pour une valeur $n$ donnée, le programme effectue $n-2$ itérations en temps constant. La complexité en temps du programme est donc $\mathcal{O}(n)$.

(2.7) (a) Pour montrer que le programme est correct, on doit montrer que lorsque le programme termine son exécution, les valeurs des variables sont correctes. On souhaite alors établir la validité des 2 triplets suivants :

$$\{n \geq 0, 0 \leq m \leq n, m > (n-m), \texttt{fact} = 1\}$$

```
for (int i = n; i > m; i--){
    fact *= i;
    fact /= (i-m);
}
```

$$\{\texttt{fact} = \frac{n!}{m!(n-m)!}\}$$

et

$$\{n \geq 0, 0 \leq m \leq n, m \leq (n-m)\texttt{fact} = 1\}$$

```
for (int i = n; i > (n-m); i--){
    fact *= i;
    fact /= (i-(n-m));
}
```

$$\{\texttt{fact} = \frac{n!}{m!(n-m)!}\}$$

En décomposant la 1ère boucle `for`, on obtient :

$$\{n \geq 0, 0 \leq m \leq n, m > (n-m)\texttt{fact} = 1, i = n\}$$

```
while (i > m){
    fact *= i;
    fact /= (i-m);
    i--;
}
```

$$\{\texttt{fact} = \frac{n!}{m!(n-m)!}\}$$

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 :

$$\begin{aligned} I : & n \geq 0 \\ & \text{et } 0 \leq m \leq n \\ & \text{et } m > (n - m) \\ & \text{et } m \leq i \leq n \\ & \text{et } \texttt{fact} = \frac{\prod_{i<j\leq n} j}{\prod_{i<j\leq n}(j-m)} \end{aligned}$$

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = n$, ainsi que `fact` $= 1$. On a bien que $\frac{\prod_{n<j\leq n} j}{\prod_{n<j\leq n}(j-m)} = 1$.
- Pour chaque itération de boucle, on a le triplet :

$$\{I, i > m\}$$

```
fact *= i;
fact /= (i-m);
i--;
```

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement $i$ et $i'$, ainsi que `fact` et `fact'` la valeur des variables $i$ et `fact` avant et après l'itération concernée.
  - On a $i' = i - 1$. Des contraintes $i = n$ et $i > m$, on déduit $m \leq i \leq n$.
  - On a $\texttt{fact'} = \texttt{fact} * i/(i-m) = \frac{\prod_{i-1<j\leq n} j}{\prod_{i-1<j\leq n}(j-m)} = \frac{\prod_{i'<j\leq n} j}{\prod_{i<j\leq n}(j-m)}$, ce qui vérifie la postcondition.
- En fin de boucle, on a $\{I, i \leq m\}$, ce qui implique $i = m$. On a alors bien que $\texttt{fact} = \frac{n!}{m!(n-m)!} = \frac{\prod_{m<j\leq n} j}{\prod_{m<j\leq n}(j-m)}$.

En décomposant la 2ème boucle `for`, on obtient :

$$\{n \geq 0, 0 \leq m \leq n, m > (n-m)\texttt{fact} = 1, i = n\}$$

```
while (i > (n-m)){
    fact *= i;
    fact /= (i-(n-m));
    i--;
}
```

$$\{\texttt{fact} = \frac{n!}{m!(n-m)!}\}$$

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 :

$$\begin{aligned} I : & n \geq 0 \\ & \text{et } 0 \leq m \leq n \\ & \text{et } m \leq (n-m) \\ & \text{et } (n-m) \leq i \leq n \\ & \text{et } \texttt{fact} = \frac{\prod_{i<j\leq n} j}{\prod_{i<j\leq n}(j-(n-m))} \end{aligned}$$

Montrons maintenant que cet invariant est valide.

- Initialement, on a $i = n$, ainsi que $\texttt{fact} = 1$. On a bien que $\frac{\prod_{n<j\leq n} j}{\prod_{n<j\leq n}(j-(n-m))} = 1$.
- Pour chaque itération de boucle, on a le triplet :

$$\{I, i > n-m\}$$

```
fact *= i;
fact /= ((i-m);
i--;
```

$$\{I\}$$

  Montrons que ce triplet est valide, en notant respectivement $i$ et $i'$, ainsi que `fact` et `fact'` la valeur des variables $i$ et `fact` avant et après l'itération concernée.
  - On a $i' = i - 1$. Des contraintes $i = n$ et $i > (n-m)$, on déduit $(n-m) \leq i \leq n$.
  - On a $\texttt{fact'} = \texttt{fact} * i/(i-(n-m)) = \frac{\prod_{i-1<j\leq n} j}{\prod_{i-1<j\leq n}(j-(n-m))} = \frac{\prod_{i'<j\leq n} j}{\prod_{i<j\leq n}(j-(n-m))}$, ce qui vérifie la postcondition.
- En fin de boucle, on a $\{I, i \leq n-m\}$, ce qui implique $i = n-m$. On a alors bien que $\texttt{fact} = \frac{n!}{m!(n-m)!} = \frac{\prod_{n-m<j\leq n} j}{\prod_{n-m<j\leq n}(j-(n-m))}$.

Il reste à démontrer que l'exécution du programme se termine toujours. Il faut donc trouver un variant qui est un entier non négatif dont la valeur décroît strictement à chaque itération de la 1ère boucle :

$$v = i - m$$

et la 2ème boucle :

$$v = i - (n - m)$$

(b) Pour une valeur $n$ et $m$ données, le programme effectue $n - m$ ou $n - (n - m)$ itérations en temps constants. La complexité en temps du programme est donc $\mathcal{O}(n)$ (car $n \geq m$).