

# Conception Formelle

2018-2019

TP noté: Quelques algorithmes de tableaux.

Vincent Penelle

- Faites des brouillons, et, une fois les deux premières heures passées, aidez-vous de frama-c.
- Sur papier, les définitions des prédicats/fonctions peuvent être données en logique naturelle (mais en précisant les labels lorsque nécessaire). On ne demande pas de justification particulière des prédicats.
- Sur papier, annoncez clairement les invariants/variants de boucles avant de prouver chaque partie de la preuve séparément (après/pendant/avant la boucle). Les preuves de logiques du premier ordre pourront être éludées (mais justes), mais pas les calculs de WP. On pourra cependant éluder les calculs de WP identiques (ou presque) à des calculs précédents.
- Pour les deux derniers exercices, la preuve sur papier est facultative, mais appréciée (elle est aussi a priori plus longue).
- On préférera des preuves incomplètes sur papier et un code partiellement annoté à pas de rendu papier du tout et des codes entièrement annotés.
- On préférera peu d'exercices bien fait que tous mal faits.
- Deadline : mercredi 12h15 (copie papier + code moodle).

**Exercice 1: Swap** On considère la fonction `swapArray` suivante.

```
void swapArray(int *t, int i, int n, int a, int b){
1. if(a<i || a>=n || b<i || b>=n) return;
2. else{
3.     int aux = t[a];
4.     t[a] = t[b];
5.     t[b] = aux;
6.}}
```

1. Donnez un prédicat `unchangedTab{L,M}(int *tab, int *tab2, integer i, integer n)` qui est vrai si et seulement si `tab` et `tab2` sont identiques entre `i` et `n-1` inclus (`tab` étant évalué au label `L`, `tab2` au label `M`).

**SOLUTION.**  $\forall \text{int } k; i \leq k < n \Rightarrow \text{at}(\text{tab}[k], L) == \text{at}(\text{tab2}[k], M);$

2. Donnez un prédicat `wrongRangedSwap{L,M}(int *t, int *s, integer i, integer n, integer j, integer k)` qui décrit le comportement de la fonction `swap` quand `j` et `k` ne sont pas des indices permettant de faire un `swap` (i.e., en dehors des bornes `i` et `n` du tableau). Dans ce cas, `swap` ne fait rien (i.e., `t` et `s` sont identiques).

**SOLUTION.**  $(j < i \vee k < i \vee j \geq n \vee k \geq n) \Rightarrow \text{unchangedTab}\{L,M\}(t,s,i,n);$

3. Donnez un prédicat `swapInArrayAux{L,M}(int *t, int *s, integer i, integer n, integer j, integer k)` qui décrit le comportement de la fonction `swap` lorsque  $i \leq j \leq k < n$ .

SOLUTION.  $i \leq j \leq k < n \Rightarrow (\text{unchangedTab}\{L,M\}(t,s,i,j) \wedge \text{unchangedTab}\{L,M\}(t,s,j+1,k) \wedge \text{unchangedTab}\{L,M\}(t,s,k+1,n) \wedge \backslash\text{at}(t[j],L) == \backslash\text{at}(s[k],M) \wedge \backslash\text{at}(t[k],L) == \backslash\text{at}(s[j],M));$

4. En utilisant les deux prédicats précédent, donnez un prédicat `swapInArray{L,M}(int *t, int *s, integer i, integer n, integer j, integer k)` qui décrit le comportement de la fonction `swap` quelques soient les valeurs de  $i, n, j$  et  $k$ .

SOLUTION. `swapInArrayAux{L,M}(t,s,i,n,j,k)  $\wedge$  swapInArrayAux{L,M}(t,s,i,n,k,j)  $\wedge$  wrongRangedSwap{L,M}(t,s,i,n,j,k);`

5. Démontrez (sur papier) que le code donné vérifie les post-conditions `/*@ ensures swapInArrayAux {Pre,Post} (t,t,i,n,j,k)*/`, et `/*@ ensures wrongRangedSwap {Pre,Post} (t,t,i,n,j,k)*/`. Vous aurez besoin de manipulez l'expression logique du prédicat (mais contenant `unchangedTab`). **On admettra que  $\text{WP}(x=t, \text{Pred}\{L\}) = \text{Pred}\{L'\}$  si  $x$  n'est pas une variable apparaissant dans  $\text{Pred}$  ( $L$  et  $L'$  correspondant au label après et avant l'instruction  $x=t$ ).** Vous garderez le prédicat `unchangedTab` tel quel dans les preuves (et `unchangedTab{L,L}(t,t,i,n)` est vrai). Pourquoi est-ce suffisant pour prouver le contrat total?

SOLUTION. On pose  $\phi = \text{wrongRangeSwap}$  et  $T = a < i \parallel b < i \parallel a \geq n \parallel b \geq n$ .

$$\begin{aligned}
& \text{WP}(1 \rightarrow 6, \phi\{\text{Pre}, \text{Post}\}(t, t, i, n, a, b)) \\
= & T \Rightarrow \text{WP}(\text{return}, \phi\{\text{Pre}, \text{Post}\}(t, t, i, n, a, b)) \\
& \wedge \neg T \Rightarrow \text{WP}(3 \rightarrow 5, \phi\{\text{Pre}, \text{Post}\}(t, t, i, n, a, b)) \\
= & T \Rightarrow (T \Rightarrow \text{unchangedTab}\{1, 1\}(t, t, i, n)) \\
& \wedge \neg T \Rightarrow (\text{WP}(3 \rightarrow 4, T \Rightarrow \text{unchangedTab}\{\text{Pre}, 5\}(t, t, i, b)) \\
& \quad \wedge \text{unchangedTab}\{\text{Pre}, 5\}(t, t, b+1, n) \wedge \text{aux} == \backslash\text{at}(t[b], \text{Pre})) \\
= & T \Rightarrow (T \Rightarrow \top) \\
& \wedge \neg T \Rightarrow (\text{WP}(3, T \Rightarrow \text{unchangedTab}\{\text{Pre}, 4\}(t, t, i, a)) \\
& \quad \wedge \text{unchangedTab}\{\text{Pre}, 4\}(t, t, a+1, b) \wedge \text{unchangedTab}\{\text{Pre}, 4\}(t, t, b+1, n)) \\
& \quad \wedge \text{aux} == \backslash\text{at}(t[b], \text{Pre}) \wedge t[b] == \backslash\text{at}(t[a], \text{Pre})) \\
= & \top \\
& \wedge \neg T \Rightarrow (T \Rightarrow \text{unchangedTab}\{\text{Pre}, 3\}(t, t, i, a)) \\
& \quad \wedge \text{unchangedTab}\{\text{Pre}, 3\}(t, t, a+1, b) \wedge \text{unchangedTab}\{\text{Pre}, 3\}(t, t, b+1, n)) \\
& \quad \wedge t[a] == \backslash\text{at}(t[b], \text{Pre}) \wedge t[b] == \backslash\text{at}(t[a], \text{Pre})) \\
= & \top \wedge \neg T \Rightarrow (T \Rightarrow \perp) \\
= & \top \wedge \top = \top
\end{aligned}$$

On pose  $\psi = \text{swapInArrayAux}$  et  $T = a < i \parallel b < i \parallel a \geq n \parallel b \geq n$ .

```

WP(1->6,  $\psi$ {Pre,Post}(t,t,i,n,a,b))
= T  $\Rightarrow$  WP(return,  $\psi$ {Pre,Post}(t,t,i,n,a,b))
 $\wedge$   $\neg$ T  $\Rightarrow$  WP(3->5,  $\psi$ {Pre,Post}(t,t,i,n,a,b))
= T  $\Rightarrow$  ( $i \geq a \geq b < n \Rightarrow \psi'$ {1,1}(t,t,i,n,a,b))
 $\wedge$   $\neg$ T  $\Rightarrow$  (WP(3->4, T  $\Rightarrow$  unchangedTab{Pre,5}(t,t,i,a)
 $\wedge$  unchangedTab{Pre,5}(t,t,a+1,b)  $\wedge$  unchangedTab{Pre,5}(t,t,b+1,n))
 $\wedge$  aux == \at(t[a],Pre)  $\wedge$  t[a] == \at(t[b],Pre))
= (T  $\wedge$   $i \geq a \geq b < n$ )  $\Rightarrow$   $\psi'$ {1,1}(t,t,i,n,a,b))
 $\wedge$   $\neg$ T  $\Rightarrow$  (WP(3,t  $\Rightarrow$  unchangedTab{Pre,4}(t,t,i,a)
 $\wedge$  unchangedTab{Pre,4}(t,t,a+1,b)  $\wedge$  unchangedTab{Pre,4}(t,t,b+1,n))
 $\wedge$  aux == \at(t[a],Pre)  $\wedge$  t[b] == \at(t[b],Pre))
=  $\perp \Rightarrow \psi'$ {1,1}(t,t,i,n,a,b))
 $\wedge$   $\neg$ T  $\Rightarrow$  (T  $\Rightarrow$  unchangedTab{Pre,3}(t,t,i,a)
 $\wedge$  unchangedTab{Pre,3}(t,t,a+1,b)  $\wedge$  unchangedTab{Pre,3}(t,t,b+1,n))
 $\wedge$  t[a] == \at(t[a],Pre)  $\wedge$  t[b] == \at(t[b],Pre))
= T  $\wedge$   $\neg$ T  $\Rightarrow$  T)
= T  $\wedge$  T = T

```

C'est suffisant, puisque l'autre preuve pour `swapInArrayAux` sera identique à la précédente, en inversant les rôles de `a` et `b`.

- Vérifiez que `frama-c` arrive à démontrer `/*@ ensures swapInArray {Pre,Post}(t,s,i,n,j,k)*/`, et ajoutez la clause `assigns` vérifiée par cette fonction, ainsi que la clause `requires` nécessaires pour valider les gardes RTE.

**SOLUTION.** `assigns t[a],t[b];`  
`requires \valid(t + (i .. n-1));`

**Exercice 2: Minimum** On considère la fonction `minimum`.

```

int minimum(int *t, int i, int n){
1. if(n<=i) return n-1;
2. else{
3.     int min = i;
4.     int k = i+1;
5.     while(k<n){
6.         if(t[k]<t[min]) min = k;
7.         k++;
8.     }
9.     return min;
10.}

```

- Donnez un prédicat `minimum(int *t, integer i, integer n, integer min)` qui est vrai si et seulement si `min` est l'indice minimum du tableau `t` entre les indices `i` et `n-1` (inclus).

**SOLUTION.** `i  $\leq$  min < n  $\wedge$   $\forall$  integer k; i  $\geq$  k < n  $\Rightarrow$  t[min]  $\leq$  t[k];`

- Prouvez (sur papier) que le code donné vérifie la post-condition `/*@ ensures i<n ==> minimum(t,i,n,\result)*/`. Vous aurez besoin de deux invariants de boucles, un parlant de `k` (et permettant de connaître la valeur de `k` à la fin de la boucle), et un permettant de démontrer la post-condition. Vous devrez (dans la preuve de l'invariant correspondant) développer l'expression de `minimum`, mais pas trop.

**SOLUTION.**

$$\begin{aligned}
& \text{WP}(1 \rightarrow 10, i < n \Rightarrow \text{minimum}(t, i, n, \backslash \text{result})) \\
= & n \geq i \Rightarrow i < n \Rightarrow \text{minimum}(t, i, n, n-1) \\
& \neg(n \geq i) \Rightarrow \text{WP}(3 \rightarrow 8, i < n \Rightarrow \text{minimum}(t, i, n, \text{min})) \\
= & \top \wedge i < n \Rightarrow \text{WP}(3 \rightarrow 4, \text{WP}(5 \rightarrow 8, i < n \Rightarrow \text{minimum}(t, i, n, \text{min})))
\end{aligned}$$

Pour trouver  $\text{WP}(5 \rightarrow 8, i < n \Rightarrow \text{minimum}(t, i, n, \text{min}))$ , il faut déterminer des invariants de boucle.

On pose  $I_1 = i+1 \leq k \leq n$ , et  $I_2 = \text{minimum}(t, i, k, \text{min})$ .

On a  $k \geq n \wedge i+1 \leq k \leq n \wedge \text{minimum}(t, i, k, \text{min})$  implique  $i < n \wedge \text{minimum}(t, i, n, \text{min})$ , ce qui implique bien  $i < n \Rightarrow \text{minimum}(t, i, n, \text{min})$ .

Montrons maintenant que  $I_1$  et  $I_2$  sont des invariants de boucle.

$$\begin{aligned}
& \text{WP}(6;7, i+1 \leq k \leq n) \\
= & \text{WP}(6, i+1 \leq k+1 \leq n) \\
= & t[k] < t[\text{min}] \Rightarrow \text{WP}(\text{min} = k, i+1 \leq k+1 \leq n) \\
& \wedge t[k] \geq t[\text{min}] \Rightarrow \text{WP}(\text{skip}, i+1 \leq k+1 \leq n) \\
= & t[k] < t[\text{min}] \Rightarrow i+1 \leq k+1 \leq n \\
& \wedge t[k] \geq t[\text{min}] \Rightarrow i+1 \leq k+1 \leq n \\
= & i+1 \leq k+1 \leq n.
\end{aligned}$$

Ce qui est bien impliqué par  $k < n$  et  $I_1$ .

$$\begin{aligned}
& \text{WP}(6;7, \text{minimum}(t, i, k, \text{min})) \\
= & \text{WP}(6, \text{minimum}(t, i, k+1, \text{min})) \\
= & t[k] < t[\text{min}] \Rightarrow \text{WP}(\text{min} = k, \text{minimum}(t, i, k+1, \text{min})) \\
& \wedge t[k] \geq t[\text{min}] \Rightarrow \text{WP}(\text{skip}, \text{minimum}(t, i, k+1, \text{min})) \\
= & t[k] < t[\text{min}] \Rightarrow \text{minimum}(t, i, k+1, k) \\
& \wedge t[k] \geq t[\text{min}] \Rightarrow \text{minimum}(t, i, k+1, \text{min})
\end{aligned}$$

Il reste à montrer ceci est bien impliqué par  $k < n$ ,  $I_1$  et  $I_2$ .

Pour la première partie, si on a  $\text{minimum}(t, i, k, \text{min})$  et  $t[k] < t[\text{min}]$ , on a bien  $\text{minimum}(t, i, k, k)$ , et comme  $t[k] \geq t[k]$ , on en déduit bien  $\text{minimum}(t, i, k+1, k)$ .

Pour la deuxième partie, si on a  $\text{minimum}(t, i, k, \text{min})$  et  $t[k] \geq t[\text{min}]$ , on a bien  $\text{minimum}(t, i, k+1, \text{min})$ .

On vient bien de démontrer que  $I_1$  et  $I_2$  sont des invariants de boucles.

Il reste donc à montrer qu'ils sont vrais en début de programme (à condition d'être entré dans le else, évidemment), à savoir :

$$\begin{aligned}
& i < n \Rightarrow \text{WP}(3;4, i+1 \geq k \geq n) \\
= & i < n \Rightarrow \text{WP}(3, i+1 \geq i+1 \geq n) \\
= & i < n \Rightarrow i+1 \geq n = \top \\
& i < n \Rightarrow \text{WP}(3;4, \text{minimum}(t, i, k, \text{min})) \\
= & i < n \Rightarrow \text{WP}(3, \text{minimum}(t, i, i+1, \text{min})) \\
= & i < n \Rightarrow \text{minimum}(t, i, i+1, i) = \top.
\end{aligned}$$

3. Démontrez (sur papier) également que cette fonction termine (i.e., trouvez un variant de boucle et prouvez-le).

SOLUTION. On prend  $n-k$  comme variant.

$\text{WP}(6;7, n-k) = \text{WP}(6, n-k-1) = n-k-1$ . On a bien  $n-k > n-k-1$ , donc le variant décroît à chaque tour de boucle.

$k < n \Rightarrow n-k \geq 0$ , le variant reste bien positif.

Cela suffit à démontrer la terminaison de la boucle.

4. Sur frama-c, donnez et prouvez deux post-conditions (la précédente, plus une pour  $n \leq i$ ), et donnez la clause assigns pertinente, ainsi que les éventuelles préconditions permettant de valider les gardes RTE. N'oubliez pas que vous aurez besoin

d'une clause `loop assigns`. Cette spécification pourra (ou non) faire intervenir des comportements.

```
SOLUTION. requires \valid(t + (i .. n-1));
requires n>INT_MIN;
loop assigns k,min;
```

**Exercice 3: countTab** On considère la fonction suivante renvoyant le nombre d'éléments d'un tableau égaux à une valeur donnée.

```
int countTab(int *t, int n, int val){
1. if(n<=0) return 0;
2. else{
3.     int res = 0;
4.     int i = 0;
5.     while(i<n){
6.         if(t[i] == val) res++;
7.         i++;
8.     }
9.     return res;
10.}}
```

1. On va d'abord définir, via des axiome, la fonction logic integer `countTab(int *t, integer i, integer n, int v) reads t[i .. n-1]`. Donnez un axiome disant que, pour tout tableau `t`, tout `i`, tout `n` et tout `v`, si le tableau est vide (i.e.,  $n \leq i$ ), alors le tableau ne contient aucun `v`.

SOLUTION.  $\forall$  int \*t, v, integer i,n;  $n \leq i \Rightarrow \text{countTab}(t,i,n,v) == 0$ ;

2. Donnez un axiome disant que, pour tout tableau `t`, et entiers `i,n`, et `v`, si on connaît `countTab(t,i,n-1,v)` et `t[n-1] == v`, la valeur de `countTab(t,i,n,v)` est ...

SOLUTION.  $\forall$  int \*t, v, integer i,n;  $i < n \Rightarrow t[n-1] == v \Rightarrow \text{countTab}(t,i,n,v) == \text{countTab}(t,i,n-1,v) + 1$ ;

3. Donnez un axiome disant que, pour tout tableau `t`, et entiers `i,n`, et `v`, si on connaît `countTab(t,i,n-1,v)` et `t[n-1] != v`, la valeur de `countTab(t,i,n,v)` est ...

SOLUTION.  $\forall$  int \*t, v, integer i,n;  $i < n \Rightarrow t[n-1] \neq v \Rightarrow \text{countTab}(t,i,n,v) == \text{countTab}(t,i,n-1,v)$  ;

4. Démontrez (sur papier) que le code fourni calcule bien la fonction définie par les axiomes précédents (i.e., /\*@ ensures \result == countTab(t,0,n,val);\*/. Vous aurez besoin de deux invariants de boucles, un parlant de `i` (et permettant de connaître la valeur de `i` à la fin de la boucle), et un permettant de démontrer la post-condition.

SOLUTION.

```
WP(1->10, \result == countTab(t,0,n,val))
= n ≤ 0 ⇒ WP(return 0, \result == countTab(t,0,n,val))
  ∧ n > 0 ⇒ WP(3->9, \result == countTab(t,0,n,val))
= n ≤ 0 ⇒ 0 == countTab(t,0,n,val)
  ∧ n > 0 ⇒ WP(3->8, res == countTab(t,0,n,val))
= T ∧ n > 0 ⇒ WP(3;4, WP(5->8, res == countTab(t,0,n,val)))
```

Le `T` est obtenu grâce au premier axiome. Pour continuer la preuve, il faut déterminer des invariants de boucles qui remplaceront `WP(5->8, res == countTab(t,0,n,val)`.

On pose  $I_1 = 0 \leq i \leq n$ ,  $I_2 = res == \text{countTab}(t,0,i,val)$ .

Commençons par montrer que ces invariants permettent de démontrer la post-condition :  $n \geq i \wedge I_1 \wedge I_2$  implique que  $i == n \wedge res == \text{countTab}(t,0,i,\text{val})$ , ce qui implique bien que  $res == \text{countTab}(t,0,n,\text{val})$ .

Montrons que ce sont des invariants :

$$\begin{aligned} & \text{WP}(6;7, 0 \leq i \leq n) \\ = & \text{WP}(6, 0 \leq i+1 \leq n) \\ = & t[i] == \text{val} \Rightarrow \text{WP}(\text{res}++, 0 \leq i+1 \leq n) \\ & \wedge t[i] \neq \text{val} \Rightarrow \text{WP}(\text{skip}, 0 \leq i+1 \leq n) \\ = & 0 \leq i+1 \leq n \end{aligned}$$

On a  $i < n \wedge 0 \leq i \leq n$  implique que  $0 \leq i+1 \leq n$ .

$$\begin{aligned} & \text{WP}(6;7, res == \text{countTab}(t,0,i,\text{val})) \\ = & \text{WP}(6, res == \text{countTab}(t,0,i+1,\text{val})) \\ = & t[i] == \text{val} \Rightarrow \text{WP}(\text{res}++, res == \text{countTab}(t,0,i+1,\text{val})) \\ & \wedge t[i] \neq \text{val} \Rightarrow \text{WP}(\text{skip}, res == \text{countTab}(t,0,i+1,\text{val})) \\ = & t[i] == \text{val} \Rightarrow res+1 == \text{countTab}(t,0,i+1,\text{val}) \\ & \wedge t[i] \neq \text{val} \Rightarrow res == \text{countTab}(t,0,i+1,\text{val}) \\ = & t[i] == \text{val} \Rightarrow \text{countTab}(t,0,i,\text{val})+1 == \text{countTab}(t,0,i+1,\text{val}) \\ & \wedge t[i] \neq \text{val} \Rightarrow \text{countTab}(t,0,i,\text{val}) == \text{countTab}(t,0,i+1,\text{val}) \end{aligned}$$

La dernière ligne est obtenue en utilisant l'invariant  $I_2$ .

En utilisant de plus l'invariant  $I_1$ , on a exactement la définition des axiomes 2 et 3, ce qui permet de conclure que cette formule est bien vraie, et que  $I_2$  est bien un invariant.

Il reste à montrer que les invariants sont vrais en début de programme (pourvu que l'on soit entré dans le else), soit :

$$\begin{aligned} & n > 0 \Rightarrow \text{WP}(3;4, 0 \leq i \leq n) \\ = & n > 0 \Rightarrow \text{WP}(3, 0 \leq 0 \leq n) \\ = & n > 0 \Rightarrow 0 \leq n = \top \\ & n > 0 \Rightarrow \text{WP}(3;4, res == \text{countTab}(t,0,i,\text{val})) \\ = & n > 0 \Rightarrow \text{WP}(3, res == \text{countTab}(t,0,0,\text{val})) \\ = & n > 0 \Rightarrow 0 == \text{countTab}(t,0,0,\text{val}) \end{aligned}$$

Cette dernière formule étant vraie par le premier axiome.

5. Démontrez (sur papier) la terminaison de la fonction.

SOLUTION. On prend  $n-i$  comme variant.

$I_1$  implique que  $n-i$  est toujours positif.

$\text{WP}(6;7, n-i) = \text{WP}(6, n-i-1) = n-i-1$ . On a bien  $n-i > n-i-1$ , et donc le variant diminue bien strictement à chaque tour de boucle.

Cela suffit à démontrer la terminaison de la boucle, et donc du programme.

6. Sur frama-c, prouvez la post-condition précédente, et donnez la clause `assigns` pertinente, ainsi que les éventuelles préconditions permettant de valider les gardes RTE (pour cela, vous aurez besoin d'un invariant de boucle supplémentaire – il n'est pas demandé de prouver ce dernier sur papier). N'oubliez pas que vous aurez besoin d'une clause `loop assigns`.

```
SOLUTION. requires \valid(t+ (0 .. n-1));
assigns \nothing;
loop invariant res <= i;
loop assigns i,res;
```

**Exercice 4: countMinimum (un peu plus dur)** La fonction suivante calcule le nombre d'occurrence de l'élément minimal d'un tableau.

```
int countMinimum(int *t, int i, int n){
1. if(n<=i) return 0;
2. else{
3.     int min = i;
4.     int count = 1;
5.     int k = i+1;
6.     while(k<n){
7.         if(t[k] < t[min]){
8.             min = k;
9.             count = 0;
10.        }
11.        if(t[k] == t[min]) count++;
12.        k++;
13.    }
14.    return count;
15.}}
```

- Démontrez que le code ci-dessus vérifie /\*@ ensures n<=i ==> \result == 0;\*/.

SOLUTION.

$$\begin{aligned} & \text{WP}(1 \rightarrow 15, n \leq i \Rightarrow \backslash \text{result} == 0) \\ = & n \leq i \Rightarrow \text{WP}(\text{return } 0, n \leq i \Rightarrow \backslash \text{result} == 0) \\ & \wedge n > i \Rightarrow \text{WP}(3 \rightarrow 14, n \leq i \Rightarrow \backslash \text{result} == 0) \\ = & n \leq i \Rightarrow n \leq i \Rightarrow 0 == 0 \\ & \wedge n > i \Rightarrow n \leq i \Rightarrow \text{WP}(3 \rightarrow 14, \backslash \text{result} == 0) \end{aligned}$$

La première partie de la conjonction est vraie car  $0 = 0$ , et la deuxième partie est vraie puisque  $n > i \wedge n \leq i$  est faux.

Note : on peut sortir  $n \leq i$  de  $\text{WP}(3 \rightarrow 14, \dots)$  puisque ni  $n$  ni  $i$  ne sont modifiés par ce bloc de code.

- Écrivez une post-condition décrivant la post-condition de cette fonction quand le tableau n'est pas vide (le résultat est égal à  $\text{countTab}(t, i, n, \text{val})$  pour  $\text{val}$  tel que  $\text{minimum}(t, i, n, \text{val})$  – vous aurez besoin d'un quantificateur existentiel.).

SOLUTION. ensures  $i < n \Rightarrow \exists \text{ integer } k; i \leq k < n \wedge \text{minimum}(t, i, n, k) \wedge \backslash \text{result} == \text{countTab}(t, i, n, t[k]);$

- Démontrez que le code ci-dessus vérifie la post-condition précédente, dans le cas où  $i \leq n$ . Vous aurez besoin 4 invariants de boucles. 3 comme d'habitude (un parlant de  $k$  et permettant de déduire la valeur de  $k$  en fin de boucle, un parlant de  $\text{min}$  et un parlant de la valeur de  $\text{min}$ ) qui permettent de déduire la post-condition. Vous aurez également besoin d'un invariant supplémentaire disant que pour toute valeur plus petite que  $t[\text{min}]$ , le nombre d'élément dans le tableau de cette valeur est nul.

SOLUTION.

```

WP(1->15,i<n ⇒ ∃ integer k; i ≤ k < n ∧ minimum(t,i,n,k)
  ∧ \result == countTab(t,i,n,t[k]))
= n ≤ i ⇒ WP(return 0,i<n ⇒ ∃ integer k;
  i ≤ k < n ∧ minimum(t,i,n,k) ∧ \result == countTab(t,i,n,t[k]))
  ∧ n > i ⇒ WP(3->15,i<n ⇒ ∃ integer k;
  i ≤ k < n ∧ minimum(t,i,n,k) ∧ \result == countTab(t,i,n,t[k]))
= (n ≤ i ⇒ i<n ⇒ (∃ integer k; i ≤ k < n
  ∧ minimum(t,i,n,k) ∧ 0 == countTab(t,i,n,t[k])))
  ∧ n > i ⇒ i < n ⇒ WP(3->13,∃ integer k;
  i ≤ k < n ∧ minimum(t,i,n,k) ∧ count == countTab(t,i,n,t[k]))
= n > i ⇒ WP(3->5,WP(6->13,∃ integer k;
  i ≤ k < n ∧ minimum(t,i,n,k) ∧ count == countTab(t,i,n,t[k]))

```

La première partie de la conjonction était vraie puisque  $n \leq i \wedge i < n$  est faux.

Pour  $WP(6->13, \dots)$ , il faut déterminer et prouver des invariants de boucles.

On pose  $I_1 = i \leq \min < k \leq n$ ,  $I_2 = \text{minimum}(t,i,k,\min)$ ,  $I_3 = \text{count} == \text{countTab}(t,i,k,t[\min])$ , et  $I_4 = \forall \text{integer val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k,\text{val}) == 0$ .

Premièrement, on a bien que  $k \geq n \wedge I_1 \wedge I_2 \wedge I_3$  implique que  $\text{count} = \text{countTab}(t,i,n,\min) \wedge \text{minimum}(t,i,n,\min) \wedge i \leq \min < n$ , ce qui implique bien la post-condition.

On prouve maintenant que les invariants sont bien inductifs.

```

WP(7->12,i ≤ min < k ≤ n)
= WP(7->11,i ≤ min < k+1 ≤ n)
= WP(7->10,t[k] == t[min] ⇒ WP(count++,i ≤ min < k+1 ≤ n)
  ∧ t[k] != t[min] ⇒ i ≤ min < k+1 ≤ n)
= WP(7->10,i ≤ min < k+1 ≤ n)
= t[k] < t[min] ⇒ WP(8->10,i ≤ min < k+1 ≤ n)
  t[k] >= t[min] ⇒ i ≤ min < k+1 ≤ n
= t[k] < t[min] ⇒ i ≤ k < k+1 ≤ n

```

On a  $I_1 \wedge k < n$  qui implique bien que  $i \leq \min < k+1 \leq n$ , ce qui permet de conclure que la première implication est toujours vraie, et  $k < n$  implique bien que  $i \leq k < k+1 \leq n$ , ce qui permet de conclure que la deuxième implication est toujours vraie.

```

WP(7->12,minimum(t,i,n,min))
= WP(7->10,minimum(t,i,n,min)) :11,12 ne modifient pas la formule
= t[k] < t[min] ⇒ WP(7;8,minimum(t,i,n,min))
  ∧ t[k] ≥ t[min] ⇒ minimum(t,i,n,min)

```

Voir exo 2, c'est la même démonstration.

```

WP(7->12,count == countTab(t,i,k,t[min]))
= WP(7->11,count == countTab(t,i,k+1,t[min]))
= WP(7->10,t[k] == t[min] ⇒ count+1 == countTab(t,i,k+1,t[min])
  ∧ t[k] != t[min] ⇒ count == countTab(t,i,k+1,t[min]))
= WP(7->10,ϕ(count,t,i,n,min)) renommage
= t[k] < t[min] ⇒ WP(8->10,ϕ(count,t,i,n,min))
  ∧ t[k] ≥ t[min] ⇒ ϕ(count,t,i,n,min)
= t[k] < t[min] ⇒ ϕ(0,t,i,n,k) ∧ t[k] ≥ t[min] ⇒ ϕ(count,t,i,n,min)

```

Prenons la première partie de la conjonction :

```

t[k] < t[min] ⇒ (t[k] == t[k] ⇒ 1 == countTab(t,i,k+1,t[k])
  ∧ t[k] != t[k] ⇒ 0 == countTab(t,i,k+1,t[k]))
= t[k] < t[min] ⇒ 1 == countTab(t,i,k+1,t[k])

```



$I_4$  implique bien que  $t[k] < t[\min] \Rightarrow 0 == \text{countTab}(t,i,k,t[k])$ , et l'axiome 2 permet de déduire le résultat, puisque  $t[k] == t[k]$ .

Deuxième partie de la conjonction :

$$\begin{aligned} & t[k] \geq t[\min] \Rightarrow (t[k] == t[\min] \Rightarrow \text{count}+1 == \text{countTab}(t,i,k+1,t[\min])) \\ & \wedge t[k] != t[\min] \Rightarrow \text{count} == \text{countTab}(t,i,k+1,t[\min])) \\ = & t[k] == t[\min] \Rightarrow \text{count}+1 == \text{countTab}(t,i,k+1,t[\min]) \\ & \wedge t[k] > t[\min] \Rightarrow \text{count} == \text{countTab}(t,i,k+1,t[\min]) \end{aligned}$$

Les axiomes 2 et 3 permettent de conclure que  $I_3$  implique bien cette formule (cf exo 3).

$$\begin{aligned} & \text{WP}(7 \rightarrow 12, \forall \text{ val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k,\text{val}) == 0) \\ = & \text{WP}(7 \rightarrow 11, \forall \text{ val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k+1,\text{val}) == 0) \\ = & \text{WP}(7 \rightarrow 10, \forall \text{ val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k+1,\text{val}) == 0) \\ & (\text{count n'est pas présent dans la formule}) \\ = & t[k] < t[\min] \Rightarrow \text{WP}(8;9, \forall \text{ val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k+1,\text{val}) == 0) \\ & \wedge t[k] \geq t[\min] \Rightarrow \forall \text{ val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k+1,\text{val}) == 0 \\ = & t[k] < t[\min] \Rightarrow \forall \text{ val}; \text{val} < t[k] \Rightarrow \text{countTab}(t,i,k+1,\text{val}) == 0 \\ & \wedge t[k] \geq t[\min] \Rightarrow \forall \text{ val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k+1,\text{val}) == 0 \end{aligned}$$

L'invariant  $I_4$  et  $t[k] < t[\min]$  implique bien que  $\forall \text{ val}; \text{val} < t[k] \Rightarrow \text{countTab}(t,i,k+1,\text{val})$ , puisque  $\text{val} < t[k]$  implique  $\text{val} < t[\min]$ , et si  $\text{val} < t[k]$  alors  $\text{val} != t[k]$ , donc l'axiome 3 donne bien le résultat. Si  $t[k] \geq t[\min]$  et  $\text{val} < t[\min]$ , on a bien  $\text{val} != t[k]$ , et donc l'invariant  $I_4$  et l'axiome 3 donnent bien la partie droite de la dernière implication.

Il reste maintenant à démontrer que ces invariants sont satisfaits en début de boucle en remplaçant  $\text{WP}(6 \rightarrow 13, \dots)$  par leur conjonction dans la preuve principale.

$$\begin{aligned} & n > i \Rightarrow \text{WP}(3 \rightarrow 5, i \leq \min < k < n) \\ = & n > i \Rightarrow \text{WP}(3 \rightarrow 4, i \leq \min < i+1 < n) \\ = & n > i \Rightarrow \text{WP}(3, i \leq \min < i+1 < n) \\ = & n > i \Rightarrow i \leq i < i+1 < n = \top \\ & n > i \Rightarrow \text{WP}(3 \rightarrow 5, \text{minimum}(t,i,k,\min)) \\ = & n > i \Rightarrow \text{WP}(3 \rightarrow 4, \text{minimum}(t,i,i+1,\min)) \\ = & n > i \Rightarrow \text{WP}(3, \text{minimum}(t,i,i+1,\min)) \\ = & n > i \Rightarrow \text{minimum}(t,i,i+1,i) = \top \text{ (c'est le seul élément du tableau)} \\ & n > i \Rightarrow \text{WP}(3 \rightarrow 5, \text{count} == \text{countTab}(t,i,k,t[\min])) \\ = & n > i \Rightarrow \text{WP}(3 \rightarrow 4, \text{count} == \text{countTab}(t,i,i+1,t[\min])) \\ = & n > i \Rightarrow \text{WP}(3, 1 == \text{countTab}(t,i,i+1,t[\min])) \\ = & n > i \Rightarrow 1 == \text{countTab}(t,i,i+1,t[i]) = \top \text{ (axiomes 1 et 2)} \\ & n > i \Rightarrow \text{WP}(3 \rightarrow 5, \forall \text{ integer val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,k,\text{val}) == 0) \\ = & n > i \Rightarrow \text{WP}(3 \rightarrow 4, \forall \text{ integer val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,i+1,\text{val}) == 0) \\ = & n > i \Rightarrow \text{WP}(3, \forall \text{ integer val}; \text{val} < t[\min] \Rightarrow \text{countTab}(t,i,i+1,\text{val}) == 0) \\ = & n > i \Rightarrow \forall \text{ integer val}; \text{val} < t[i] \Rightarrow \text{countTab}(t,i,i+1,\text{val}) == 0 \\ = & \top \text{ (axiomes 1 et 2)} \end{aligned}$$

#### 4. Démontrez la terminaison de la fonction.

SOLUTION. Variant de boucle :  $n-k$ .

$n-k \geq 0$  est impliqué par  $I_1$ .

$\text{WP}(7 \rightarrow 13, n-k) = \text{WP}(7 \rightarrow 11, n-k-1) = n-k-1$  (car les deux autres instructions ne modifient pas  $k$  et  $n$ ).  $n-k > n-k-1$  pour tous  $n$  et  $k$ , ce qui démontre la décroissance stricte du variant.

- Ajoutez la clause `assigns` pertinente, ainsi que les éventuelles préconditions permettant de valider les gardes RTE (pour cela, vous aurez besoin d'un invariant de boucle supplémentaire). N'oubliez pas que vous aurez besoin d'une clause `loop assigns`.

```
SOLUTION. requires \valid(t + (i .. n-1))
requires n-i < INT_MAX;
loop invariant count ≤ k-i;
loop assigns k,min,count;
```

**Exercice 5: sortByPermutation (plus dur)** On met maintenant un peu toutes les fonctions précédentes ensemble pour un algorithme de tri.

Attention, ici, pour les preuves sur papier, vous aurez à remonter deux appels de fonctions. Pour les démontrer, vous découperez chaque formule en deux parties : celle parlant de valeurs modifiées par la fonction (dans le `assigns`), et celle parlant de valeurs invariantes. La deuxième partie devra être vraie avant l'appel de la fonction, tandis que la première devra être impliquée par la post-condition de la fonction (et vous aurez donc à démontrer que la pré-condition est vraie avant l'appel – mais ici, les pré-conditions sont  $\top$ , donc ça devrait aller).

```
void sortByPermutation(int *t, int i, int n){
1. if(n<i) return;
2. else{
3.     int j = i;
4.     while(j<n){
5.         int min = minimum(t,j,n);
6.         swapArray(t,i,n,j,min);
7.         j++;
8. }}}
```

- Donnez un prédicat `predicate sorted(int *t, integer i, integer n)` qui est vrai si et seulement si le tableau `t` est trié entre `i` et `n`.
- Démontrez que le code ci-dessus vérifie la post-condition `/*@ ensures sorted{Post}(t,i,n);*/`. Pour cela, vous aurez besoin de 3 invariants de boucles : un parlant de `j`, un parlant de la partie déjà triée du tableau (comme d'habitude), mais vous aurez aussi besoin d'un invariant indiquant que le dernier élément que la partie triée du tableau est plus petit que tous les éléments situés après lui (attention, au début de la boucle, cet élément n'existant pas, il faut, grâce à une implication bien choisie, assurer que l'invariant reste vrai).
- Donnez un prédicat `permutation{L,M}(int *t, int *s, integer i, integer n)` `reads \at(t[i .. n-1],L),\at(s[i .. n-1],M)`; qui décrit que le tableau `s` est une permutation du tableau `t`. Vous aurez besoin de deux axiomes pour ceci : un disant qu'un tableau est une permutation de lui-même; et un disant que si on a trois tableaux `t,s` et `u`, si `s` est une permutation de `t` et qu'il existe `j` et `k` tels que `swapInArray(s,u,i,n,j,n)`, alors `u` est une permutation de `t`.
- Démontrez que le code ci-dessus vérifie la post-condition `/*@ ensures permutation{Pre,Post}(t,t,i,n);*/`. Vous aurez besoin d'un invariant de boucle supplémentaire.
- Ajoutez la clause `assigns` pertinente, ainsi que les éventuelles préconditions permettant de valider les gardes RTE. N'oubliez pas que vous aurez besoin d'une clause `loop assigns`.