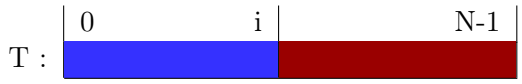
# INFO0947: Rapport Projet 1

Groupe 23: Andrew Willems, Pierre Lorenzen

# Table des matières

# 1 Description du problème

Il est demandé de filtrer un tableau d'entiers par rapport à une certaine propriété $p$.
On peut représenter le problème comme suit :




Avec la zone bleu qui concerne la zone filtrée du tableau(A), la taille de la zone filtrée(B) et tout les éléments qui s'y trouve se trouvait dans le même ordre dans le tableau initiale(C). La zone brune concerne la zone non filtrée ce qui veut dire que la zone est remplie de 0(D).

# 2 Découpe en sous-problème

Pour avoir plus de facilité à résoudre le problème nous pouvons le séparer en 2 sous-problèmes. Le premier sous-problème étant le problème principal : Filtrer le tableau par rapport à une certaine propriété $p$. Les éléments respectant cette propriété sont à l'avant du tableau et dans le même ordre que dans le tableau initiale. Les éléments ne respectant pas la propriété sont placé en bout de tableau et leur valuer est mise à 0.
Le deuxième sous-problème est la permutation de 2 éléments d'un même tableau sans changer leur valeur.

## 2.1 Relation entre les sous-problème

$SP_2 \subset SP_1$

# 3 Sous-problème 2

Nous allons commencer par définir un prédicat avec leur objets utilisés et une notation pour pouvoir décrire ce sous-problème plus facilement.

## 3.1 Spécification du prédicats

1. Objets Utilisés
   - ⋆ T : Un tableau d'entier initialisé de taille N.
   - ⋆ N $>$ 0 ($\in \mathbb{N}$)
   - ⋆ $T_0$ : Le tableau T avant modification.
2. Signature
   *Permutation(T,N,$T_0$)*
3. Spécification
   *Permutation(T,N,$T_0$)*$\equiv (\forall i, 1 \leq i < N, (\exists j, 0 \leq j < N, T_0[j] = T[i])$

## 3.2 Objets Utilisés

Ensuite, nous allons décrire les objets utilisés par le sous-problème 2.
- ⋆ T : Un tableau d'entier initialisé de taille N.
- ⋆ N $>$ 0 ($\in \mathbb{N}$)

⋆ i : est la destination de la valeur à permuter.
⋆ j : est le depart de la valeur à permuter.

### 3.3 Signature

La signature du sous-problème 2 est la suivante :

```
1        void perm(int *T, int const N, int const i, int const j);
```

### 3.4 Spécification

Ici, nous décrivons la précondition et la postcondition du sous-problème.

```
1        /**
2         * @Précond : N > 0 ∧ Tinit ∧ 0 ≤ i, j < N
3         * @Postcond :N = N_0 ∧ Permutation(T, N, T_0)
4         */
5        void perm(int *T, int const N, int const i, int const j);
```

### 3.5 Code

Dans cette section, nous écrivons le bout de code correspondant au sous-problème 2 avec ses assertions intermédiaires pour nous assurer la validiter du code.

```
1        /**
2        * @Précond : N > 0 ∧ Tinit ∧ 0 ≤ i, j < N
3        * @Postcond :N = N_0 ∧ Permutation(T, N, T_0)
4        */
5        void perm(int *T, int const N, int const i, int const j){
6            assert(T != NULL && N > 0);
7            //@Précond (>≡ N > 0 ∧ Tinit ∧ 0 ≤ i, j < N<)
8            int k = T[i];
9            //(>N > 0 ∧ Tinit ∧ 0 ≤ i, j < N ∧ k = T_0[i]<)
10           T[i]=T[j];
11           //(>N > 0 ∧ Tinit ∧ 0 ≤ i, j < N ∧ k = T_0[i] ∧ T[i] = T_0[j] ∧ T[j] = k<)
12           T[j]= k;
13           //(>N > 0 ∧ Tinit ∧ 0 ≤ i, j < N ∧ k = T[i] ∧ T[i] = T_0[j] ∧ T[j] = T_0[i]<)
14           //(>N > 0 ∧ Tinit(∀i, 1 ≤ i < N, (∃j, 0 ≤ j < N, T_0[j] = T[i])<)
15           //(>N = N_0 ∧ Permutation(T, N, T_0)<)
16           // Postcond
17       }
```

## 4 Sous-problème 1

Nous allons commencer par définir quelques prédicats avec leur objets utilisés et quelque notations pour pouvoir décrire ce sous-problème plus facilement.

### 4.1 Spécification des prédicats

#### 4.1.1 Spécification du prédicat B

1. Objets Utilisés
   ⋆ T : Un tableau d'entier initialisé de taille N.
   ⋆ N $> 0$ ($\in \mathbb{N}$)
   ⋆ p : Une certaine propriété.

2. Signature
   *TailleZoneFiltree(T,N,p)*
3. Spécification
   *TailleZoneFiltree(T,N,p)*$\equiv \#i, 0 \leq i < N, p(T[i])$

#### 4.1.2 Spécification du prédicat A

1. Objets Utilisés
   - ⋆ T : Un tableau d'entier initialisé de taille N.
   - ⋆ taille : taille de la zone filtrée.
   - ⋆ p : Une certaine propriété.
2. Signature
   *ZoneFiltree(T,p,taille)*
3. Spécification
   *ZoneFiltree(T,p,taille)*$\equiv \forall i, 0 \leq i < taille, p(T[i])$

#### 4.1.3 Spécification du prédicat C

1. Objets Utilisés
   - ⋆ $T_0$ : Le tableau T avant modification.
   - ⋆ T : Un tableau d'entier initialisé de taille N.
   - ⋆ $N > 0$ ($\in \mathbb{N}$)
   - ⋆ taille : taille de la zone filtrée.
2. Signature
   *LienTableau*$(T_0, T, N, taille)$
3. Spécification
   *LienTableau*$(T_0, T, N, taille) \equiv (\forall i, 1 \leq i < taille, (\exists j, 0 \leq j < N, T_0[j] = T[i]) \wedge (\exists k, 0 \leq k < j, T_0[k] = T[i-1]))$

#### 4.1.4 Spécification du prédicat D

1. Objets Utilisés
   - ⋆ T : Un tableau d'entier initialisé de taille N.
   - ⋆ $N > 0$ ($\in \mathbb{N}$)
   - ⋆ taille : taille de la zone filtrée.
2. Signature
   *ZoneNonFiltree(T,N,taille)*
3. Spécification
   *ZoneNonFiltree(T,N,taille)*$\equiv \forall i, taille < i < N, T[i] = 0$

### 4.2 Objets utilisés

Ensuite, nous allons décrire les objets utilisés par le sous-problème 1.
- ⋆ T : Un tableau d'entier initialisé de taille N.
- ⋆ $N > 0$ ($\in \mathbb{N}$)

## 4.3 Signature

La signature du sous-problème 1 est la suivante :

```
1        int filtrer(int*T, const int N);
```

## 4.4 Spécification

Ici, nous décrivons la précondition et la postcondition du sous-problème.

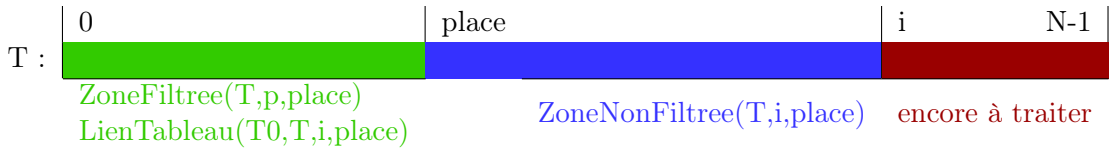
```
1        /**
2         * @Précond : N > 0 ∧ Tinit
3         * @Postcond :N = N0∧taille = TailleZoneFiltree(T, N, p)∧ZoneFiltree(T, p, taille)∧LienTableau(T0, T, N, taille
ZoneNonFiltree(T, N, taille)
4         */
5        int filtrer(int*T, const int N);
```

## 4.5 Invariant

Vu que ce sous-problème contient une boucle, il nous faut avoir un invariant graphique et formel.

### 4.5.1 Invariant graphique

Avec la spécification des prédicats ci dessus on peut trouver un invariant graphique.




Avec $place = TailleZoneFiltree(T, i, p)$.

### 4.5.2 Invariant formel

De l'invariant graphique on peut en dériver l'invariant formel :
Inv $\equiv N = N_0 \wedge 0 \leq i \leq N-1 \wedge place = TailleZoneFiltree(T, i, p) \wedge ZoneFiltree(T, p, place) \wedge LienTableau(T_0, T, i, place) \wedge ZoneNonFiltree(T, i, place)$

## 4.6 Initialisation

L'initialisation avant l'entrée dans la boucle est la suivante :

```
1        i = 0;
2        taille = 0;
```

## 4.7 Gardien de boucle

Grâce à l'invariant formel on peut trouver le gardien de boucle.

$\neg B \equiv i \geq N$

## 4.8 Corps de boucle

Dans le corps de la boucle il y a 3 étapes : la progression, la fonction de terminaison et la conclusion. On va détailler ces étapes ci-dessous.

### 4.8.1 Progression

Pour la progression, il faut se demander ce qu'il faut faire dans la boucle pour arriver à la post condition.

```
                if(test(T[i])){
                    perm(T,N,i,taille);
                    taille++;
                    i++;
                }
                else{
                    T[i] = 0;
                    i++;
                }
```

### 4.8.2 Fonction de terminaison

Pour la fonction de terminaison, on doit se demander pour quelle valeur de i on sort de la boucle.

$$F = N - i$$

### 4.8.3 Conclusion

Pour la conlusion, on doit se demander ce qu'il reste à faire après être sorti de la boucle. ce qui veut dire :

$$Inv \wedge \neg B? \equiv Post$$

Si on regarde l'invariant formel si on mei i qui est égale à N on à bien la postcondition.

## 4.9 Code

Dans cette section, nous écrivons le bout de code correspondant au sous-problème 1 avec ses assertions intermédiaires pour nous assurer la validiter du code.

```
        /**
         * @Précond : N > 0 ∧ Tinit
         * @Postcond :N = N0∧taille = TailleZoneFiltree(T, N, p)∧ZoneFiltree(T, p, taille)∧LienTableau(T0, T, N, taille)∧
ZoneNonFiltree(T, N, taille)
         */
        int filtrer(int*T, const int N){
            assert(T != NULL && N > 0);
            //@Précond (>≡ N > 0 ∧ Tinit<)
            int i = 0;
            int taille = 0;
            //(>N > 0 ∧ Tinit ∧ i = 0 ∧ taille = 0<)
            //(>N > 0 ∧ Tinit ∧ i = 0 ∧ taille = TailleZoneFiltree(T, i, p)<)
            while(i < N){
                //(>N > 0 ∧ Tinit ∧ i < N ∧ taille = TailleZoneFiltree(T, i, p)<)
                if(test(T[i])){
                    perm(T,N,i,taille);
```

```
16                          //(>N > 0∧Tinit∧i < N∧taille−1 = TailleZoneFiltree(T, i−1, p)∧ZoneFiltree(T, p, taille−
    1) ∧ LienTableau(T0, T, i − 1, taille − 1) ∧ ZoneNonFiltree(T, N, taille − 1)<)
17                          taille++;
18                          i++;
19                          //(>N > 0∧Tinit∧i < N∧taille = TailleZoneFiltree(T, i, p)∧ZoneFiltree(T, p, taille)∧
    LienTableau(T0, T, i, taille) ∧ ZoneNonFiltree(T, N, taille)<)
20                      }
21                      else{
22                          T[i] = 0;
23                          //(>N > 0∧Tinit∧i < N∧taille = TailleZoneFiltree(T, i−1, p)∧ZoneFiltree(T, p, taille)∧
    LienTableau(T0, T, i − 1, taille) ∧ ZoneNonFiltree(T, N, taille)<)
24                          i++;
25                          //(>N > 0∧Tinit∧i < N∧taille = TailleZoneFiltree(T, i, p)∧ZoneFiltree(T, p, taille)∧
    LienTableau(T0, T, i, taille) ∧ ZoneNonFiltree(T, N, taille)<)
26                      }
27                  }
28                  //(>N > 0 ∧ Tinit ∧ i = N ∧ taille = TailleZoneFiltree(T, i, p) ∧ ZoneFiltree(T, p, taille) ∧
    LienTableau(T0, T, i, taille) ∧ ZoneNonFiltree(T, N, taille)<)
29                  //(>N > 0∧Tinit∧taille = TailleZoneFiltree(T, N, p)∧ZoneFiltree(T, p, taille)∧LienTableau(T0, T, N, taille)∧
    ZoneNonFiltree(T, N, taille)<)
30                  //Postcond
31          }
```

## 5 Code Complet

Après avoir fait le code de chauqe sous-problème, il ne reste qu'a les mettre ensemble pour avoir le code complet comme ci-dessous :

```
1           void perm(int *T, int const N, int const i, int const j){
2               assert(T != NULL && N > 0);
3               int k = T[i];
4               T[i]=T[j];
5               T[j]= k;
6           }
7
8           int filtrer(int*T, const int N){
9               assert(T != NULL && N > 0);
10              int i = 0;
11              int taille = 0;
12
13              while(i < N){
14                  if(test(T[i])){
15                      perm(T,N,i,taille);
16                      taille++;
17                      i++;
18                  }
19                  else{
20                      T[i] = 0;
21                      i++;
22                  }
23              }
24          }
```