Tous les documents papiers sont autorisés. Les documents électroniques sont interdits, les ordinateurs personnels, tablettes, téléphones, montres connectés, écouteurs, et autre appareils intelligents doivent être rangés dans les sacs. Seul l'usage des ordinateurs de l'école est autorisé.

Consignes: Le sujet consiste à spécifier des fonctions à l'aide de la version en ligne de Why3. Afin de vous éviter de recopier le code manuellement, voici un lien vers un document Why3 contenant tout le code déjà écrit.

Exercice 1 (Tour de chauffe – **5 points**).

1. **(3 points)** On considère le programme itératif suivant, pour incrémenter tous les éléments d'un tableau

```
void incrementerTableau (int tab[], int n){
    for(int i = 0; i < n; i++){
        tab[i] = tab[i] + 1;
    }
}
```

   Ecrire les pré-conditions de la fonction, qui spécifient la taille du tableau, un variant pour montrer que la boucle termine, et un invariant permettant de prouver que tous les accès au tableau sont corrects.

2. **(2 point)** On considère maintenant une version récursive du même programme

```
void incrementerTableauRec (int tab[], int n){
    if (n == 0){}
    else{
        incrementerTableauRec (tab, n-1);
        tab[n] = tab[n]+1;
    }
}
```

   Ecrire les pré-conditions de la fonction, qui spécifient la taille du tableau, un variant pour montrer que la fonction termine.

Exercice 2 (Somme des deux éléments maximaux – **11 points**). Dans cet exercice, on travaille sur deux programmes calculent la somme des deux plus grands éléments d'un tableau:

1. **Version itérative:**

```
int sommeMax (int tab[], int n){
    int max;
```

```
    int second;
    if(tab[0] <= tab[1]){
        second = tab[0];
        max = tab[1];
    }
    else {
        max = tab[0];
        second = tab[1];
    }
    for(int i = 2; i < n; i++){
        if (max <= tab[i]){
            second = max;
            max = tab[i];
        } else if (second < tab [i]){
            second = tab[i];
        }
    }
    return max + second;
}
```

(a) **(1 point)** Donner les pré-conditions de la fonction qui assurent que l'entier donné est bien la longueur du tableau.

(b) **(1 point)** Donner le variant pour montrer que la boucle termine

(c) **(2 point)** Donner la précondition et l'invariant qui garantissent que tous les accès au tableau sont corrects.

(d) **(2 point)** Donner une post-condition qui garantit que le résultat est bien la somme de deux éléments du tableau. C'est-à-dire, qu'il existe deux positions différentes telles que le résultat de la fonction est la somme des valeurs contenues dans le tableau à ces position.

2. **Version récursive:** On considère maintenant la version récursive du même programme:

```
int sommeMaxRecAux (int tab[], int n, int max, int second){
    if (n == 0){
        return max + second;
    }
    if (max <= tab[n-1]){
        second = max;
        max = tab[n-1];
    } else if (second < tab[n-1]){
        second = tab[n-1];
    }
    return sommeMaxRecAux(tab, n-1, max, second);
}
```

```
int sommeMaxRec (int tab[], int n){
    int max;
    int second;
    if (tab[n-1] >= tab[n-2]){
        max = tab[n-1];
        second = tab[n-2];
    }
    else{
        max = tab[n-2];
        second = tab[n-1];
    }
    return sommeMaxRecAux(tab, n-2, max, second);
}
```

(a) **(1 point)** Donner une précondition pour la fonction `sommeMaxRec` qui garantit que l'entier donné est bien la longueur du tableau, et une précondition pour la fonction `sommeMaxRecAux` qui garantit que l'entier donné est bien dans les bornes du tableau.

(b) **(1 point)** Donner un variant pour la fonction `sommeMaxRecAux`.

(c) **(1 point)** Donner une précondition pour la fonction `sommeMaxRec` qui garantit que le tableau a au moins deux entrées.

(d) **(2 points)** Donner une post-condition pour la fonction `sommeMaxRec` qui garantit que le résultat est bien la somme de deux éléments du tableau. Donner une pré-condition et une post-condition pour la fonction `sommeMaxRecAux` qui permettent de prouver la post-condition de la fonction `sommeMaxRec`. La précondition de la fonction `sommeMaxRecAux` devra porter sur l'existence de positions supérieures à `n`.

Exercice 3 (Somme des deux éléments maximaux, suite – **4 points**).
Dans l'exercice précédent, on a donné toutes les conditions pour que la somme de deux éléments d'un tableau n'ait pas de mauvais accès au tableau et termine, et on a montré que le résultat était toujours la somme de deux éléments du tableau. Dans cet exercice, notre but est de montrer qu'il s'agit de la somme maximale. C'est-à-dire que étant deux positions différentes dans le tableau, la somme des valeurs contenues à ces positions est toujours plus petite que le résultat.

1. **(1 point)** Donner une post-condition pour les fonctions `sommeMax` et `sommeMaxRec` qui garantit que pour toute paire de positions différentes dans le tableau, la somme des valeurs contenues à ces positions est toujours plus petite que le résultat.

2. **(1.5 point)** Donner un invariant pour `sommeMax` permettant de prouver cette post-condition. L'invariant doit ressembler à la post-condition en question. **(Attention)**: L'invariant tout seul n'est pas suffisant, car

Why3 ne sait pas déterminer qu'il est bien invariant. Pour cela, il faut le combiner avec un invariant secondaire, qui garantit que la variable `max` contient le maximum parmi les premier éléments du tableau. Ecrire cet invariant secondaire.

3. **(1.5 point)** Donner une pré-condition et une post-condition pour `sommeMaxRecAux` permettant de prouver la post-condition pour `sommeMaxRec`. La post-condition doit être similaire à celle de `sommeMaxRec`, tandis que la pré-condition doit porter sur les éléments du tableau contenues à des positions supérieures à `n`. **(Attention)**: Comme à la question précédente, la pré-condition seule n'est pas suffisante, il faut le combiner avec une pré-condition secondaire, qui garantit que la variable `max` contient le maximum parmi les éléments du tableau de position supérieure à `n`. Ecrire cette pré-condition secondaire.