TD3 : Preuves de terminaison et de correction

Exercice 1 (Division enclidienne) On considère l'algorithme suivante :

Algorithme 1 Division euclidienne

entrée : entiers naturels a et b, avec $b > 0$
sortie : quotient q, reste r

1 : $q \leftarrow 0$; $w \leftarrow b$; $r \leftarrow a$
2 : **tant que** $w \leq r$ **faire**
3 : $\quad w \leftarrow 2 * w$
4 : **fin tant que**
5 : **tant que** $w \neq b$ **faire**
6 : $\quad q \leftarrow 2 * q$
7 : $\quad w \leftarrow w/2$ /* division entre entiers */
8 : $\quad$ **si** $w \leq r$ **alors**
9 : $\quad\quad r \leftarrow r - w$
10 : $\quad\quad q \leftarrow q + 1$
11 : $\quad$ **fin si**
12 : **fin tant que**
13 : renvoie (q, r)

(je ne mets pas la fonction `C` parce que vous ne savez pas encore comment renvoyer un couple de valeurs en `C`)

1. Faites tourner la fonction à la main sur les valeurs `(10,4)`, puis sur les valeurs `(15,5)` pour trouver ce que fait cette fonction.
2. Montrer que l'appel à la fonction se termine quelles que soient les entrées (qui vérifient les pré-conditions indiquées), en utilisant des variants de boucle.
3. Montrer que la fonction calcule effectivement ce que votre intuition vous a indiqué lors de la question 1 (si vous êtes en panne d'intuition : la fonction est documentée), en utilisant un invariant de boucle.

Exercice 2 (Tri bulle) On donne la fonction suivante qui exécute le *tri bulle*.

```c
/**
 * entree : un tableau d'entiers et sa taille
 * sortie : le tableau est trie en place
 */
void tri_bulle(int *tab, int n){
  bool permutation = true;
  while(permutation){ // si le dernier passage a fait au moins une modification
    permutation = false;
    for(int i=0; i<n-1; i++){
      if(tab[i] > tab[i+1]){ // si ce n'est pas dans le bon ordre, on inverse
        int tmp = tab[i];
        tab[i] = tab[i+1];
        tab[i+1] = tmp;
        permutation = true;
      }
    }
    n = n - 1;
  }
}
```

1. Montrer que l'exécution de la fonction `tri_bulle` se termine quelle que soit l'entrée (en supposant que `n` est bien la longueur du tableau `tab`).
2. Montrer qu'à la fin de l'exécution de la fonction `tri_bulle`, les données du tableau sont triées. Pour cela, vous pourrez trouver un invariant qui explicite le nombre de données à leur place finale après chaque itération de la boucle `while`.

Exercice 3 (Produit) On donne la fonction suivante :

```
/** a, b : entiers positifs
    resultat : a*b */
int multiplication(int a, int b){
  int m = 0;

  assert(a>=0);
  assert(b>=0);

  while(a>0){
    if(a%2 == 1){
      m = m+b;
    }
    a = a/2;
    b = b*2;
  }

  return m;
}
```

1. Faites tourner la fonction `multiplication` à la main sur les valeurs 13 et 15 et constater qu'elle semble faire ce qu'elle prétend faire.
2. Montrer que l'appel à la fonction `multiplication` se termine quelles que soient les entrées, en utilisant un variant de boucle.
3. Montrer que le fonction calcule effectivement le produit des deux arguments qui lui sont transmis, en utilsant un invariant de boucle.

Exercice 4 (Exponentiation rapide) On considère la fonction d'exponentiation rapide :

```
/** a : entier
     n : entier positif ou nul
     sortie : a puissance n
*/
int puissance(int a, int n){
  int p = 1;
  while (n > 0) {
    if (n%2 == 1){
      p = p * a;
    }
    a = a * a;
    n = n / 2;
  }

  return p;
}
```

1. Dérouler cette fonction sur l'entrée $(2,5)$.
2. Donner un variant de boucle qui permet de conclure que cette fonction se termine.
3. Montrer grâce à un invariant de boucle que cette fonction fait bien ce qu'on attend d'elle.