TD #3 — Correction

Le but de ce TD est de prouver la correction d'un certain nombre d'algorithmes (écrits en C) plus ou moins complexes. Les preuves de correction pourront utiliser des invariants *de boucle* qui satisfont les conditions suivantes : $\mathcal{I}$ est un *invariant (utile) de la boucle* `while (cond) { ... }` si :

— $\mathcal{I}$ est vrai avant la première itération de la boucle
— Si $\mathcal{I}$ et la condition d'entrée dans la boucle sont vrais, alors $\mathcal{I}$ est vrai à la fin d'une itération du corps de boucle *si cette itération termine*

Si la boucle ne contient pas de `break`, ceci permet de déduire que si l'on atteint le point du programme suivant immédiatement la boucle, alors $\mathcal{I}$ et la négation de `cond` y sont vrais.
Un (tout petit) peu plus formellement, soit `S` la suite d'instruction d'un corps de boucle `while` *sans* `break`, si l'on a la propriété : « si $\mathcal{I} \wedge \mathcal{C}$ sont vrais avant d'exécuter `S`, alors $\mathcal{I}$ est vrai après exécution de `S` », alors si $\mathcal{I}$ est vrai avant `while (C) S`, on a $\mathcal{I} \wedge \neg\mathcal{C}$ après l'exécution de la boucle toute entière.

Exercice 1. *Élément maximum dans un tableau*

On se donne la fonction C suivante :

```c
int max(size_t n, int a[n]) {
    assert(n > 0); // requis par le langage
    int m = a[0];
    size_t i = 1; // pour simplifier la rédaction
                  // de l'invariant
    for (; i < n; i++)
    {
        if (a[i] > m) {
            m = a[i];
        }
    }
    return m;
}
```

1. Prouvez sa terminaison.
2. Prouvez sa correction relativement aux spécifications informelles suivantes : « `max` renvoie un élément maximum du tableau `a`, c'est à dire un élément `m` présent dans `a` et tel qu'il n'existe aucun autre élément `x` de `a` avec `x > m` ».

Exercice 2. *Recherche d'un élément dans un tableau*

On se donne la fonction C suivante :

```c
int search(int n, int a[n], int e) {
    assert(n > 0); // requis par le langage
    int i = 0; // pour simplifier la rédaction
               // de l'invariant
    for (; i < n; i++)
    {
        if (a[i] == e) {
            return i;
        }
    }
    return -1;
}
```

1. Prouvez sa terminaison.
2. Prouvez sa correction relativement aux spécifications informelles suivantes : « `search` renvoie `-1` si aucun élément de `a` n'est égal à `e`, et sinon renvoie une valeur `i` telle que `a[i] == e` ».

Exercice 3. *Recherche d'un élément dans un tableau trié, par dichotomie* ⋆⋆

On se donne la fonction C suivante :

```c
int bsearch(int n, int a[n], int e) {
    int b = 0;
    int t = n - 1;

    while (b < t) {
        int m = b + (t - b)/2;
        if (e == a[m]) {
            return m;
        }
        if (e < a[m]) {
            t = m - 1;
        }
        else {
            b = m + 1;
        }
    }

    if ((b > t) || (a[b] != e)) {
        return -1;
    }
    return b;
}
```

1. Prouvez sa terminaison
2. Prouvez sa correction relativement aux spécifications informelles suivantes : « soit `a` un tableau d'éléments triés par ordre croissant, `bsearch` renvoie `-1` si aucun élément de `a` n'est égal à `e`, et sinon renvoie une valeur `i` telle que `a[i] == e` ».

Exercice 4. *Exponentiation lente*

Prouvez la terminaison et la correction de votre fonction `sloexp` du TD#2.

Exercice 5. *Exponentiation rapide*

Prouvez la terminaison et la correction de votre fonction `fastexp` du TD#2.

Teaser : prouvez que la fonction ci-dessous calcule le PGCD de a et b, quand $a \geq b$.

```c
uint64_t bgcd(uint64_t a, uint64_t b) {
    uint64_t r = 1;
    while (a > 0 && b > 0) {
        while ((a % 2 == 0) && (b % 2) == 0) {
            r *= 2;
            a /= 2;
            b /= 2;
        }
        while (a % 2 == 0) {
            a /= 2;
        }
        while (b % 2 == 0) {
            b /= 2;
        }
        if (a < b) {
            b = b - a;
        }
        else {
            uint64_t t = b;
            b = a - b;
            a = t;
        }
    }
    return a * r;
}
```