Informatique: TD1

MP2I Lycée Pierre de Fermat

Exercice 0.

Correction faite en classe

Exercice 1. *Correction faite en classe*

Exercice 2. On considère l'algorithme suivant :

Algorithme 1 : Quinquonce

```
Entrée(s) : n ∈ ℕ
Sortie(s) : Rien
N ← n + 1;
tant que n > 0 faire
    si n == N alors
        n ← n − 1;
    sinon
        N ← N − 1;
```

Correction question 1.

Correction question 2. Aucun des deux n'est a priori un variant de boucle, car à chaque passage de boucle, soit n soit N reste constant.

Correction question 3. Montrons que "$N \geq n$" est un invariant de boucle.

— En entrée de boucle, $N = n + 1 \geq n$: la propriété est vérifiée.

— Plaçons nous au début d'un passage de boucle, et supposons $N \geq n$. On note n, N' les valeurs de n et N à la fin du passage. On distingue deux cas :

  — $n = N$. Alors, $n' = n - 1$ et $N' = N$. Par hypothèse, $N \geq n$ donc $N' = N \geq n > n - 1 = n'$.

  — $n! = N$. Par hypothèse, $N \geq n$, et donc $N > n$. Or, $n' = n$ et $N' = N - 1$. Donc, $N' = N - 1 > n - 1$, et donc $N' \geq n = n'$.

  Dans les deux cas, on a bien $N' \geq n'$.

Donc, $N \geq n$ est bien un invariant de boucle, et est donc vraie à chaque fois que l'on teste la condition de boucle.

Correction question 4. Montrons que $n + N$ est un variant de boucle.

— n et N sont entières au début de l'algorithme, et ne se voient assigner que des valeurs entières, donc $n + N$ est toujours un entier.

— En entrée de la boucle, $N = n + 1$ et $n \in \mathbb{N}$, il est donc clair que $n + N \geq 0$. De plus, si l'on exécute un passage de la boucle while, alors $n > 0$, et donc $n + N \geq 2$. Cependant, au cours d'un passage, soit n soit N est décrémenté de 1, mais pas les deux. Donc, à la fin du passage, $n + N$ a été décrémenté de 1 et est donc supérieur à 1 (et donc à 0).

Donc, $n + N$ est entière, positive et décroît strictement à chaque boucle. Donc, c'est bien un variant de boucle : cet algorithme termine.

Exercice 3.

Dans cette correction, on note $a \wedge b$ le PGCD de a et b.

Correction question 1. La fonction suivante implémente l'algorithme d'Euclide :

```c
/*
  Renvoie le PGCD de a et b.
  Précondition: a, b positifs et non simultanément nuls
*/
int pgcd(int a, int b){
  assert(a >= 0 && b >= 0);
  assert(a != 0 | b != 0);
  int r, s, t;
  r = a;
  s = b;
  while (r != 0){
    t = s%r;
    s = r;
    r = t;
  }
  return s;
}
```

Correction question 2. Vous savez le faire

Correction question 3. Montrons que r est un variant de boucle.

— r est de type `int`, donc ne prend que des valeurs entières.
— Au cours de l'algorithme, à chaque passage de la boucle, r reçoit systématiquement le reste d'une division euclidienne par l'ancienne valeur de r, autrement dit r décroît strictement mais reste positive.

Donc, r est bien un variant de boucle.

Correction question 4. Montrons que la propriété suivante est un invariant de boucle :

$$\mathcal{P} \quad : \quad r \wedge s = a \wedge b$$

En TD je vous ai donné une propriété du PGCD qui permet de prouver ça, ici on fait la preuve en revenant à la définition du PGCD (ce qui revient à démontrer la propriété admise dans l'énoncé) !

— En entrée de boucle, $r = a$ et $s = b$ donc P est vraie.
— Supposons P vraie au début d'un passage de boucle. Puisque l'on passe dans la boucle, on a donc $r > 0$. Notons r', s' les valeurs de r et s à la fin du passage. On a $r' = s\%r$ et $s' = r$. On veut montrer :

$$r' \wedge s' = r \wedge s = a \wedge b$$

On écrit la division euclidienne de s par r :

$$s = qr + r'$$

avec $q \in \mathbb{N}$.

Donc, $r' = s - qr$. Il nous suffit donc de montrer que $r \wedge s = (s - qr) \wedge r$. Nous allons procéder en montrant que chacun des deux divise l'autre. Par définition du PGCD, il nous suffit de montrer deux choses :

— Si $d \in \mathbb{N}$ divise r et s, alors d divise $s - qr$ et r. Cela garantira que tous les diviseurs communs de r et s sont aussi des diviseurs communs de $s - qr$ et r, et en particulier que $r \wedge s$ divise $(s - qr) \wedge r$.

— Si $d \in \mathbb{N}$ divise $s - qr$ et r, alors d divise r et s. Cela garantira de même que $(s - qr) \wedge r$ divise $r \wedge s$.

Prouvons le premier point (le deuxième se fait de manière analogue). Soit $d \in \mathbb{N}$ tel que d divise r et s. Alors, d divise toute combinaison entière de r et s, et en particulier d divise $s - qr$.

Ainsi, $r \wedge s = (s - qr) \wedge r = r' \wedge s' = a \wedge b$, et donc la propriété est toujours vraie à la fin du passage de boucle.

P est bien un invariant de boucle. En particulier, en sortie de boucle, on a $r = 0$ et donc $a \wedge b = 0 \wedge s = s$: l'algorithme renvoie bien $a \wedge b$.

Exercice 4.

Exercice 5.

Exercice 6.

Exercice 7.

Exercice 8.

Correction question 1. Un algorithme simple est :

Algorithme 2 : Racine entière

Entrée(s) : $x \in \mathbb{R}^*$
Sortie(s) : La racine entière de x

$i \leftarrow 0$;
tant que $i^2 \leq x$ **faire**
  $i++$;
retourner ***i-1***

Correction question 2. Montrons directement que pour $i \in \mathbb{N}$, on a $x_{i+1} \geq \sqrt{a}$. On a :

$$\begin{array}{rcl} x_{i+1} - \sqrt{a} & = & \frac{1}{2}(x_i + \frac{a}{x_i} - 2\sqrt{a}) \\ & = & \frac{1}{2}(\sqrt{x_i} - \frac{\sqrt{a}}{\sqrt{x_i}})^2 \\ & \geq & 0 \end{array}$$

Donc $x_{i+1} \geq \sqrt{a}$. De plus, $x_0 \geq \sqrt{a}$.

Montrons maintenant la décroissance. Soit $i \in \mathbb{N}$. On a :

$$\begin{array}{rcl} x_{i+1} - x_i & = & \frac{1}{2}(x_i + \frac{a}{x_i} - 2x_i) \\ & = & \frac{1}{2}(\frac{a}{x_i} - x_i) \end{array}$$

Or, $x_i \geq \sqrt{a}$, donc $a/x_i \leq a/\sqrt{a} = \sqrt{a} \leq x_i$, donc $x_{i+1} - x_i \leq 0$ car $x_i > 0$. La suite $(x_i)_{i\in\mathbb{N}}$ est bien décroissante.

Correction question 3. Pour $i \in \mathbb{N}$ on a $x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i})$. En faisant tendre i vers $+\infty$, on obtient l'équation $l = \frac{1}{2}(l + \frac{a}{l})$, soit $l^2 = a$. Comme les x_i sont positifs, l est aussi positive, et donc $l = \sqrt{a}$.

Correction question 4. *Il y avait une coquille dans l'énoncé : il fallait montrer* $\leq \frac{|x_i - \sqrt{a}|^2}{2x_i}$.

Soit $i \in \mathbb{N}$. On a $x_{i+1} \geq \sqrt{a}$ donc $|x_{i+1} - \sqrt{a}| = x_{i+1} - \sqrt{a}$. On a :

$$\begin{array}{rcl} |x_{i+1} - \sqrt{a}| & = & x_{i+1} - \sqrt{a} \\ & = & \frac{1}{2}(x_i + \frac{a}{x_i} - 2\sqrt{a}) \\ & = & \frac{1}{2x_i}(x_i^2 + a - 2x_i\sqrt{a}) \\ & = & \frac{|x_i - \sqrt{a}|^2}{2x_i} \\ & \leq & \frac{|x_i - \sqrt{a}|^2}{2\sqrt{a}} \text{ car } x_i > \sqrt{a}. \end{array}$$

Correction question 5. Comme $a > 1$, on a $|x_{i+1} - \sqrt{a}| < |x_i - \sqrt{a}|^2$. On en déduit par récurrence la formule demandée.

Correction question 6. $x_0 = \lceil \sqrt{a} \rceil$ donc $x_0 - \sqrt{a} \leq 1$. Donc :

$$\begin{array}{rcl} |x_1 - \sqrt{a}| & \leq & \frac{1^2}{2} \\ |x_2 - \sqrt{a}| & \leq & \frac{|x_1 - \sqrt{a}|^2}{2} \leq \frac{1}{8} \\ |x_3 - \sqrt{a}| & \leq & \frac{|x_2 - \sqrt{a}|^2}{2} \leq \frac{1}{128} < \frac{1}{10} \end{array}$$

Donc $n_0 = 3$ suffit.

Correction question 7. On peut prendre $N = 3 + \log_2(n)$: 3 coups pour arriver à une précision de $\frac{1}{10}$, c'est à dire d'une décimale, puis $\log_2(n)$ coups pour arriver à une précision de n décimales. En pratique la convergence est même bien plus rapide car on a été très grossiers dans les majorations !

Exercice 9.

Correction question 1.

Correction question 2. On trouve dans l'ordre : 0, 4, 4, 5.

Correction question 3. Cette fonction semble calculer et renvoyer la racine entière de a. On pourrait commenter et mettre une assertion comme suit :

```c
// Renvoie la racine entière de a entier positif
int racine_entiere(int a){
  assert(a >= 0);
  ...
```

Correction question 4. Notons $V = a - s + 2i + 1$. Montrons que V est bien un variant de boucle.

— V est composée uniquement avec des variables et constantes entières, donc est entière.

— V est positive au début de la boucle car alors $s = 1, i = 0$ donc $V = a - 1 + 1 = a \geq 0$. De plus, si au début d'un passage V est positive, alors par condition de boucle on a aussi $s \leq a$. Notons a', s', i' les valeurs de a, s, i à la fin du passage. On a :

  — $a' = a$

  — $i' = i + 1$

  — $s' = s + 2i' + 1 = s + 2i + 3$

  Donc, en notant $V' = a' - s' + 2i' + 1$, on a :

$$V' = a - (s + 2i' + 1) + 2i' + 1 = a - s$$

  Or, $0 \geq a - s$ par condition de boucle. Donc V est bien positive à la fin du passage.

— En reprenant les notations du point précédent, on a $V' = a - s = V - (2i + 1)$. Donc $V' < V$: V décroît strictement à chaque passage.

Finalement, V est bien un variant de boucle, donc la boucle while termine, et donc la fonction aussi.

Correction question 5. Il faut montrer qu'à la fin de l'exécution, on a $i^2 \leq a < (i + 1)^2$. De plus, l'idée est que s contient à chaque étape le carré suivant. On peut donc proposer l'invariant suivant :

$$s = (i + 1)^2 \text{ et } i^2 \leq a$$

Montrons que c'est bien un invariant de la boucle while :

— En entrée de boucle, $s = 1$ et $i = 0$, et de plus $a \geq 0$ par précondition de la fonction. Donc, les deux parties de l'invariant sont vérifiées.

— Supposons la propriété vraie au début d'un passage de boucle, montrons qu'elle l'est toujours à la fin. On a donc $s = (i+1)^2$ et $i^2 \leq a$. En reprenant les mêmes notations qu'à la question précédente, on a

  — $s' = s + 2i + 3$

  — $i' = i + 1$

  — $a' = a$

  Donc $s' = s + 2i + 3 = (i + 1)^2 + 2i + 3 = i^2 + 4i + 4 = (i + 2)^2$: la première partie de l'invariant est vérifiée.

  De plus, $i'^2 = (i + 1)^2 = s^2$. Or, par condition de boucle, on a $s \leq a$, donc $i'^2 \leq a$: la deuxième partie de l'invariant est également vérifiée.

Finalement, la propriété énoncée est bien un invariant, et en particulier, lorsque l'on sort de la boucle c'est que $a < s$, on a donc, en combinant cela avec l'invariant :

$$i^2 \leq a < s = (i + 1)^2$$

D'où la correction de l'algorithme.

Correction question 6. L'autre algorithme qu'on a vu est le suivant (directement implémenté en C) :

```
int racine_entiere(int n){
  int i = 0;
  while (i*i <= n){
      i = i + 1;
  }
    return i-1;
}
```

En pratique, les deux fonctions font de l'ordre de $\sqrt{n}$ tours de boucles. Cependant, dans la fonction vue en cours, à chaque passage on doit calculer une multiplication : $i \times i$. Dans la fonction de l'exercice, on fait uniquement des additions, et une multiplication par 2, mais comme les entiers sont stockés en binaire, la multiplication par 2 est presque gratuite (cf chapitre 2 du cours). Cette méthode est donc plus efficace !

Notons qu'il existe une méthode encore plus rapide, n'effectuant qu'un nombre logarithmique d'opérations, basée sur la **dichotomie** !