Informatique: TD1

MP2I Lycée Pierre de Fermat

Exercice 0.

Exponentiation rapide

On reprend l'exponentiation rapide :

Algorithme 1 : Exp. rapide

Entrée(s) : $x \in \mathbb{R}$, $n \in \mathbb{N}$
Sortie(s) : x^n

```
1 N ← n;
2 r ← 1;
3 X ← x;
4 tant que N > 0 faire
5 |   si N%2 = 1 alors
6 |   |   r ← X × r ;
7 |   N ← N//2;
8 |   X ← X × X ;
9 retourner r
```

Q1. Simulez l'exécution de l'algorithme pour :

1. $x = 3, n = 3$
2. $x = 2, n = 9$

Q2. Trouvez un variant de boucle permettant de montrer la terminaison de l'algorithme.

Q3. Énoncez la correction de l'algorithme avec une phrase de la forme "à la fin de l'exécution, [Formule impliquant les entrées et les sorties]".

Q4. Formulez un invariant de boucle reliant les valeurs de r, X, N et x^n.

Q5. Prouvez que la propriété est bien un invariant de boucle.

Q6. Déduisez-en que l'algorithme est correct.

Exercice 1.

Fonction mystère

Considérons la fonction suivante :

```
1 int mystere(int a, int b){
2   int x = 0;
3   int c = 0;
4   while (c < b){
5     c = c + 1;
6     x = x + a;
7   }
8   return x;
9 }
```

Q1. Simulez l'exécution de cette fonction pour les entrées suivantes :

1. $a = 7, b = 4$
2. $a = -4, b = 3$
3. $a = 0, b = 2$
4. $a = 5, b = -7$

Que fait cette fonction ? Comment la commenteriez-vous ?

Q2. Trouvez un variant de boucle, et montrez la terminaison de cette fonction

Q3. Trouvez un invariant de boucle permettant de prouver la correction de cette fonction.

Exercice 2.

Descente en quinconce

Les invariants de boucle ne servent pas uniquement à prouver la correction d'algorithmes. Étudions un exemple où l'on utilise un invariant pour prouver une propriété servant ensuite à prouver la terminaison.

On considère l'algorithme suivant :

Algorithme 2 : Quinquonce

Entrée(s) : $n \in \mathbb{N}$
Sortie(s) : Rien

```
1 N ← n + 1;
2 tant que n > 0 faire
3     si n == N alors
4         n ← n − 1;
5     sinon
6         N ← N − 1;
```

Q1. Simulez l'exécution de cet algorithme pour $n = 6$.

Q2. n est-il un variant de boucle ? Et N ?

Q3. Montrez que $N \geq n$ est un **invariant** de boucle.

Q4. Montrez que $n + N$ est un variant de boucle.

Exercice 3. *Euclide*

On rappelle l'algorithme d'Euclide pour le calcul du plus grand diviseur commun :

Algorithme 3 : Algorithme d'Euclide

Entrée(s) : a et b deux entiers positifs, avec $(a, b) \neq (0, 0)$
Sortie(s) : le PGCD de a et b

```
1 r ← a;
2 s ← b;
3 tant que r ≠ 0 faire
4 |   t ← le reste de la division euclidienne de s par r;
5 |   s ← r;
6 |   r ← t;
7 retourner s
```

Q1. Écrivez une fonction `pgcd` en C implémentant cet algorithme. N'oubliez pas les commentaires et les assertions.

Q2. Simulez l'exécution de l'algorithme d'Euclide pour

1. $a = 385, b = 121$
2. $a = 17, b = 26$
3. $a = 26, b = 17$
4. $a = 8, b = 0$

Q3. Trouvez un variant de boucle, et montrez la terminaison de cette fonction.

Q4. En trouvant un invariant de boucle adéquat, montrez la correction de cet algorithme. On pourra utiliser le fait que pour $x, y \in \mathbb{N}$, si z est le reste de la division euclidienne de x par y, alors $\mathbf{PGCD}(x, y) = \mathbf{PGCD}(y, z)$.

Q5. Comment modifier simplement cet algorithme pour qu'il renvoie le bon résultat pour $a, b \in \mathbb{Z}$ et pas seulement pour $a, b \in \mathbb{N}$?

Exercice 4. *Recherche de minimum*

Étant donné n nombres $x_0, \dots x_{n-1} \in \mathbb{R}$, on souhaite calculer efficacement leur minimum, et même trouver l'indice $i_0 \in [\![0, n-1]\!]$ tel que $x_{i_0} = \min\{x_i | i \in [\![0, n-1]\!]\}$.

Q1. Proposez un algorithme en pseudo-code répondant à ce problème :

Algorithme 4 : `indice_min`

Entrée(s) : $n \in \mathbb{N}^*$, $(x_0, \dots, x_{n-1}) \in \mathbb{R}^n$
Sortie(s) : $i_0 \in [\![0, n-1]\!]$ tel que $x_{i_0} = \min\{x_i | i \in [\![0, n-1]\!]\}$

Vous pourrez vous appuyer sur l'idée suivante : utiliser une boucle pour parcourir chacun des x_i, et stocker l'indice du plus petit vu pour l'instant dans une variable.

Q2. Exhiber un variant de boucle et montrer la terminaison de votre programme.

Q3. Exhiber un invariant de boucle adéquat et montrer la correction de votre programme.

Exercice 5.

Des invariants à l'algorithme

On cherche à implémenter un algorithme simple de division euclidienne. Nous allons utiliser cette occasion pour voir un autre usage des invariants et variants de boucles : nous allons retrouver l'algorithme **à partir** des invariants/variants.
Étant donnés $a, b \in \mathbb{N}$ avec $(a, b) \neq (0, 0)$, on souhaite calculer un couple $(q, r) \in \mathbb{N}^2$ tel que $a = qb + r$ et $0 \leq r < b$. Le code à trou suivant utilise des variables q et r qui vérifient toujours $a = qb + r$ et $0 \leq r$ tout au long de l'algorithme, mais pas forcément $r < b$:

Algorithme 5 : Division euclidienne

Entrée(s) : $a, b \in \mathbb{N}$ avec $(a, b) \neq (0, 0)$
Sortie(s) : $(q, r) \in \mathbb{N}^2$ tel que $a = qb + r$ et $0 \leq r < b$

```
1 q ← ...;
2 r ← ...;
  // Invariant A: 0 ≤ r
  // Invariant B: a = qb + r
  // Variant de boucle: r
3 tant que ... faire
4     ...;
5 retourner (q, r)
```

Q1. Remplir les initialisations de q et r afin que les invariants A et B soient vérifiés en entrée de boucle.

Q2. Quelle condition de boucle mettre afin qu'en fin d'exécution, le couple (q, r) corresponde à la spécification demandée ?

Q3. Compléter le corps de la boucle de façon à ce que r soit un variant, et que les invariants A et B se conservent au cours d'un passage.

Exercice 6.

Graphe de flot de contrôle

Dessinez les graphes de flot de contrôle des fonctions suivantes, et donnez pour chacune une jeu de test couvrant toutes les arêtes du graphe.

Q1.

```c
int f(int x){
  if (x%32 == 0){
    printf("A\n");
  }
  else if (x%8 == 0){
    printf("B\n");
  }
  else if (x%2 == 0){
    printf("C\n");
  } else {
    printf("D\n");
  }
}
```

Q2.

```c
float g(float x, int n){
  float y = x;
  int j;
  for (int i = 1; i <= n; i++){
    j = 1;
    while (j < i){
      j = j * (j+1);
      y = y/2;
    }
    y = y + j;
  }
  return y;
}
```

Q3.

```c
float g(int x, int n){
  assert(n>=0);
  if (x <= 0){
    for(int i = 0; i < n; i++){
      printf("%d\n", i);
    }
  } else {
    for(int i = 0; i < n*x; i++){
      if (i%x == 0){
        printf("%d\n", i);
      }
    }
  }
  return 14.22;
}
```

Exercice 7. *Terminaison et correction en récursif*

Voyons une méthode pour montrer la terminaison et la correction d'une fonction récursive. On souhaite écrire une fonction calculant le n-ème terme d'un suite arithmético-géométrique.

Rappel : Une suite arithmético géométrique satisfait une équation de la forme :

$$u_n = au_{n-1} + b \tag{1}$$

avec $a, b \in \mathbb{R}$. Lorsque $b = 0$ on obtient une suite géométrique, lorsque $a = 1$ on obtient une suite arithmétique. Considérons la fonction suivante :

```
1  float arith_geom(float a, float b, int n, float u0){
2    /* Renvoie le n-eme terme de la suite arithmético-géométrique
3       de paramètres a, b, de premier terme u0 */
4    assert(n>=0);
5    if (n == 0){
6      return u0;
7    } else {
8        float precedent = arith_geom(a, b, n-1, u0);
9      return a * precedent + b;
10   }
11 }
```

Q1. Soient $a, b, u_0 \in \mathbb{R}$. Montrez par récurrence que pour tout $n \in \mathbb{N}$, l'appel `arith_geom(a, b, n, u0)` termine en un temps fini.

Q2. Montrez par récurrence sur $n \in \mathbb{N}$ que `arith_geom(a, b, n, u0)` renvoie bien le n-ème terme de la suite arithmético-géométrique de paramètres a, b, de premier terme u_0.

Entraînons-nous maintenant sur un autre exemple. Soient $a \in \mathbb{R}^{+*}$ et $b \in R$. On considère l'algorithme récursif suivant :

Algorithme 6 : temps_de_vol

Entrée(s) : x réel

si $x < 0$ **alors**
  retourner 0
sinon
  retourner $1 +$ ***temps_de_vol***$(a * \sqrt{x} + b)$

Le but est de montrer sa terminaison.

Q3. A quelle condition sur a, b a-t'on que pour tout $x \geq 0$, l'appel récursif causé par `temps_de_vol(x)` se fait sur une quantité strictement inférieure à x ?

Q4. Afin de pouvoir borner le nombre d'appels récursifs, on veut que ceux-ci se fassent non-seulement sur des quantités décroissantes, mais en plus avec un écart minimal fixé. Soit $\epsilon > 0$. Sous quelle condition sur a, b, ϵ a t'on que $a * \sqrt{x} + b < x - \epsilon$?

Q5. Pour $x \in R$ positif et $\epsilon > 0$ satisfiant la condition précédente, bornez le nombre d'appels récursifs effectués lors de l'exécution de l'algorithme sur x, en fonction de ϵ.

Exercice 8.

Algorithme de Héron

Le calcul de la racine carréee d'un nombre réel est une opération utilisée dans de très nombreux programmes (simulation physique, graphismes, etc...). Il existe plusieurs algorithmes d'approximation de la racine carrée, dont certains très complexes et très efficaces. Étudions un de ceux les plus simples : l'algorithme de Héron. Il consiste à faire un choix initial estimé proche de la racine, puis à affiner ce choix itérativement :

Algorithme 7 : Algorithme de Héron

Entrée(s) : $a > 1$ un réel, $n \in \mathbb{N}$

Sortie(s) : Une approximation de $\sqrt{a}$ sur n décimales

```
1 N ←// à déterminer
2 x ← 1 + racine_entiere(a);
3 pour i = 0 à N − 1 faire
4     x ← 1/2 (x + a/x);
5 retourner x
```

Soit $a \in \mathbb{R}^+$. Soit $(x_i)_{i\in\mathbb{N}}$ la suite définie par :

$$x_0 = \lceil \sqrt{a} \rceil \tag{2}$$

$$x_{i+1} = \frac{x_i + \frac{a}{x_i}}{2} \tag{3}$$

Q1. Donnez un algorithme permettant de calculer la racine entière, c'est à dire, étant donné un réel positif x, de donner le plus grand entier k tel que $k^2 <= x$.

Q2. Montrez que $x_i \geq \sqrt{a}$ pour $i \in \mathbb{N}$, et que la suite $(x_i)_{i\in\mathbb{N}}$ est décroissante.

Q3. On en déduit que la suite $(x_i)_i$ converge. Soit l sa limite. Montrez $l = \sqrt{a}$.

Donc, pour tout $n \in \mathbb{N}$, à partir d'un certain rang, x_i approche $\sqrt{a}$ à moins de $\frac{1}{10^n}$, i.e. avec n décimales correctes. Il reste à déterminer le nombre d'itérations à faire. Pour cela on étudie la vitesse de convergence de la suite.

Q4. Montrez que $|x_{i+1} - \sqrt{a}| \leq \frac{|x_i - \sqrt{a}|^2}{2\sqrt{x_i}} \leq \frac{|x_i - \sqrt{a}|^2}{2\sqrt{a}}$

Q5. On suppose qu'il existe $n_0 \in \mathbb{N}$ tel que $|x_{n_0} - \sqrt{a}| < \frac{1}{10}$, i.e. tel que la première décimale correspond. Montrez que pour $k \in \mathbb{N}$, $|x_{n_0+k} - \sqrt{a}| < \frac{1}{10^{2^{k+1}}}$.

Cette inégalité signifie qu'à partir de n_0, à chaque itération le nombre de décimales correctes double. La dernière étape est de majorer n_0'

Q6. Montrez $|x_0 - \sqrt{a}| \leq 1$. Déduisez-en une majoration sur le nombre d'itérations n_0 nécessaires pour avoir $|x_{n_0} - \sqrt{a}| < \frac{1}{10}$

Q7. Déterminez une valeur à assigner à N au début de l'algorithme garantissant la correction de la spécification donnée.

Exercice 9.

Déja-vu ?

Considérons la fonction suivante :

```
1 int mystere_0(int a){
2   int s = 1;
3   int i = 0;
4   while (s <= a){
5     i = i + 1;
6     s = s + 2*i + 1;
7   }
8   return i;
9 }
```

Q1. Pour $a = 13$, simulez l'exécution de la fonction et indiquez la valeur renvoyée. Pour cela, vous remplirez un tableau indiquant la valeur de chaque variable après l'exécution de chaque ligne de chaque passage de boucle. Le tableau a été partiellement rempli ci dessous :

passage	1			2			3			4		
ligne	4	5	6	4	5	6	4	5	6	4	5	6
a	13	13	13	13	13	13	13	13	13	13	13	13
i	0	1	1	1	2	2	2					
s	1	4	4	4								

Q2. Donnez la valeur de retour de la fonction `mystere_0` pour :

a) $a = 0$ b) $a = 16$ c) $a = 17$ d) $a = 35$

Q3. Que fait cette fonction ? Rédigez un commentaire pour la fonction expliquant son fonctionnement, et donnez une ou plusieurs assertions nécessaire au bon fonctionnement.

Q4. Montrez que $a - s + 2i + 1$ est un variant de boucle. Déduisez en la terminaison de cette fonction.

Q5. Énoncez et montrez la correction de cet algorithme en utilisant un invariant de boucle adéquat.

Q6. (Bonus) Nous avons en cours un algorithme plus simple calculant la même valeur. Rappelez le principe de cet algorithme. A votre avis, si l'on traduisait ces deux algorithmes en fonctions C, laquelle serait la plus efficace ? Pourquoi ?