UTA Année académique **2023-2024**

IGL-3

TD n0 3 : Programmation avancée et Méthodes Formelles

Programmation avancée

Exercice 1.

Ecrire une fonction qui recherche séquentiellement le maximum dans un tableau.

1. Donner la spécification de ce problème :
   a. Entrée et Sortie
   b. Rôle
   c. Précondition
   d. Postcondition
2. Ecrire en python le problème ainsi spécifié

Exercice 2.

Partie I : Prouver la correction de ce programme

1. Commencer par trouver un invariant de boucle
   a. Chercher une propriété qui vérifie la précondition, avant d'exécuter la boucle i.
   b. On rentre ensuite dans **la phase initialisation**
      i. L'invariant est-il vrai avant la première itération ?
   c. Puis la **phase conservation**
      i. L'invariant est-il maintenu vrai par une itération de la boucle ?
      ii. Pour cela supposer que l'invariant est vraie pour l'indice i et montrer qu'alors il est vrai pour l'indice i+1.
2. On rappelle qu'un algorithme se termine s'il ne boucle pas à l'infini. L'algorithme précédent termine-t-il ? Justifier votre réponse.

Partie II : Terminaison d'un algorithme avec un variant de boucle

Dans cette partie, on considère l'algorithme de recherche séquentiel d'une occurrence.

```
def appartient(v, T):
    i = 0
    trouvee = False
    while i < len(T) and trouvee == False:
        if T[i] == v:
            trouvee = True
        i = i + 1
    return trouvee
```

1. On définit la notion de variant comme suit. Un variant est une expression à valeur entière dépendant des variables impliquées dans la répétitive dont on peut démontrer que la valeur : est positive ou nulle et elle décroit au cours des itérations. Justifier que la quantité $len(T) - i$ est un variant. Pour prouver qu'un algorithme termine, il suffit de montrer qu'il ne boucle pas à l'infini.
2. En déduire que l'algorithme termine.

N.B. :

1. Toute algorithme sans appel de fonction ni répétitive termine ;

2. Toute répétitive pour itère un nombre fini de fois (en Python) ;

3. Une répétitive tant que (et les appels récursifs) peut boucler à l'infini si sa condition reste toujours vraie.

Exercice 1. On cherche à calculer la somme de deux polynômes représentés par des tableaux. Par exemple, $X^5 + 3X^4 + 5$ est représenté par le tableau 5 ; 0; 0; 0; 3; 1.
1. Écrire une spécification du problème.
2. Écrire un programme solution.
3. Prouver la correction du programme par rapport à la spécification du problème.

Exercice 2. On cherche à déterminer l'élément minimum d'un tableau.
1. Écrire une spécification du problème.
2. Écrire un programme solution.
3. Prouver la correction du programme par rapport à la spécification.

Exercice 3.
1. Écrire un programme impératif prenant en entrée un entier n et permettant de calculer la somme des n premiers entiers.
2. Prouver la correction du programme et sa terminaison

Exercice 4

On considère la fonction réalisant la division euclidienne a/b de deux entiers naturel a et b et renvoyant le quotient q et le reste r.

```
def division(a,b):
    """a est un entier naturel
    b est un entier naturel non nul
    renvoie le quotient q et le reste r de la division a/b
    a=q*b+r est un invariant de boucle et r<b """
    q=0
    r=a
    while r>b:
        q=q+1
        r=r-b
    return q,r
```

1) Prouver la terminaison de cet algorithme.
2) Chercher à montrer que ce code est correct. Conclure.
3) Proposer un exemple d'assertion soulignant le problème de correction de cette fonction.
4) Corriger ce programme et vérifier sa correction au regard des spécifications

Exercice 5

Voici un algorithme pour déterminer le plus grand élément d'une liste non vide de nombres.
Prouver sa correction

```
def maximum(L):
    i=0
    maxi=L[i]
    for i in range(1,len(L)):
        if L[i]>maxi:
            maxi=L[i]
    return maxi
```

Exercice 4. Voici deux programmes pour calculer 2^navec un entier n naturel

```
def puissance1(n):
    p=1
    for i in range(1,n+1):
        p=p*2
    return p
def puissance2(n):
    p=1
    b=2
    m=n
    while m > 0:
        m,r=m//2,m%2
        p=p*b**r
        b=b*b
    return p
```

1) Dans le 1e programme, quel est le nombre d'affectations et de multiplications effectuées dans le corps de la boucle en fonction de n ? Evaluer le niveau de complexité.

2) Prouver la terminaison du 2e programme en estimant le nombre d'itérations.

3) Prouver la correction du 2e programme en considérant l'invariant de boucle $pb^m=2^n$

4) Justifier alors que le niveau de complexité de cet algorithme est en $O(log_2(n))$

Exercice. Démontrer que l'algorithme de tri par sélection terminent.

```
def echange(T, i, j):
    """échange T[i] et T[j]"""
    temp = T[i]
    T[i] = T[j]
    T[j] = temp

def tri_par_selection(T):
    """trie le tableau T dans l'ordre croissant"""
    for i in range(len(T)):
        ind_min = i
        for j in range(i+1, len(T)):
            if T[j] < T[ind_min]:
                ind_min = j
        echange(T, i, ind_min)
```

Exercice. Démontrer que l'algorithme de tri par insertion terminent.

```
def tri_par_insertion(T):
    """trie le tableau T dans l'ordre croissant"""
    for i in range(1,len(T)):
        x = T[i]
        j = i
        while j>0 and x < T[j-1]:
            T[j] = T[j-1]
            j = j - 1
        T[j] = x
```