# TD2: algo: terminaison et validité d'un algo iteratif

correction

March 2021

## Contents



# 1 notes:

Program Conditions Algorithms or code can be documented with comments that state conditions that should be true when program execution reaches the point where the comment is located. These comments are called preconditions, invariants, or postconditions, depending on their placement relative to a code or algorithm segment.

1. A precondition(termination) is a statement placed before the segment. It must be true prior to entering the segment in order for it to work correctly. Preconditions are often placed either before loops or at the entry point of functions and procedures.

2. A postcondition is a statement placed after the end of the segment. It should be true when the execution of the segment is complete. Postconditions are often placed either after loops or at exit points of functions and procedures.

3. An invariant is a statement placed in a code segment that is repeated should be true each time the the loop execution reaches that point. Invariants are often used in a loops and recursions.

4. If an invariant is placed at the beginning of a while or do..while loop whose condition test does not change the state of the computation (that is, it has no side effects), then the invariant should also be true at the bottom of the loop.

# 2 Exercice 1:Factorielle iterative

On considere la fonction FactorielleIt definie de la maniere suivante pour calculer la factorielle pour n $\in$ N :

```
def FactorielleIt(n):
i=1; tmp=1
while i≤n:
tmp=tmp*i
i=i+1
return tmp
```

## 2.1 Question 1:Mt la terminaison de l'algo

Le corps de la boucle est composé d'instructions elementaires.

La boucle est effectué n fois.

Donc factorielle(n) se termine pour tout n $\in$ N.

## 2.2 Question 2:Que vaut tmpi en fonction de i ? En deduire un invariant de boucle et demontrer le

L'invariant est $tmp_i$=i(i-1)!=i! avec tmp=(i-1)! et i=i et $0 \leq i \leq n$.
Pour prouver cet invariant il faut prouver que cet invariant est vrai en entrant dans la boucle(ligne1) et apres chaque iteration. on commence par prouver le cas de base par induction:

Cas de base: en entrant dans la boucle on a tmp=1 et i=1

donc, $tmp_1$ = 0!x1=1 et, $0 \leq i \leq n+1$ comme on le voulait.

Puis le cas d'induction:

On suppose que pour tout n ∈ N , l'invariant tmpi est vrai avant l'iteration. donc, on suppose ici que tmpi=i(i-1)!=i! et 0 ≤ i ≤ n+1 sont vrai.

demontrons que l'hypothese est vrai apres chaque iteration,

soit les valeurs arbitraires, $tmp'_i$ la valeur de tmpi apres l'iteration, et $i'$ celle de i.

on a donc,

$tmp'_i$= tmp*i

= (i-1)!*i avec tmp = (i-1)! et i=i

= i!*(i+1) (ligne i=i+1)

=(i+1)!

= ($i'$-1)!*$i'$ (car i+1)

maintenant montrons i ≤ n+1 c'est a dire i+1 ≤ n

donc , 0 ≤ i

< $i'$ = i+1

≤ n

### 2.3 Question 3:En deduire la validite de la fonction FactorielleIt.

La sortie de la boucle + INV = validite(partial correctness) :

On a i+1 ≤ n par l'invariant de la boucle

et, i+1 > n en sortant de la boucle

donc i=n

et comme, $tmp_i$=(i-1)!*i=i! alors $tmp_n$= (n-1)!*n=n!

donc a la ligne 6 la fonction retourne bien $tmp_n$=n! ce qui demontre la validité de factorielle n.

## 3 Exercice2: somme des elts dans un tableau

On considere la fonction somme definie de la maniere suivante pour sommer les elements d'un tableau :

```
def Somme(tab):
tmp = 0
i = 0
NbElem = len(tab)
while i < NbElem :
tmp = tmp + tab[i]
i = i + 1
return tmp
```

### 3.1 Q1: terminaison

Montrer la terminaison de l'algorithme.

le corps de la boucle est composé d'instructions elementaires.

la boucle est effectué leng(tab) fois

donc, la boucle se termine pour toute valeur i $\leq$ leng(tab).

## 3.2 Q2:Invariant

input: un tableau

output: Somme(tab)=tab[0]+..tab[NbElem]=

( $\sum_{i=0}^{NbElem}$ tab[i] )*length(tab)

Soit tmpi pour i $\in \{0, , NbElem\}$ la valeur de la variable tmp aux instants suivants :

-$tmp_0$ la valeur de tmp juste avant d'entrer dans la boucle

-pour i $\in \{1, , NbElem\}$ , tmpi est la valeur de i de tmp a la fin du corps de la boucle. (juste apres l'incrementation de i). ainsi $tmp_1$ vaut tab[0].

que vaut tmpi en fonction de i? en deduire par reccurence. Step1: find the appropriate invariants for the loop

$tmp_i$=( $\sum_{j=0}^{i-1}$ tab[j] )

i $\leq$ NbElem

Step 2: prove that LI holds when entering the loop and after every iteration.

Cas de base: juste avant d'entrer dans la boucle on a:

tmp=0, i=0, et nbElem= length(tab) pour i $\in \{0, , NbElem\}$

donc,

$tmp_0$=0 et i=0 et i  n sont vrai avant le debut de la boucle comme on le voulait, donc $tmp_i$= ($\sum_{j=0}^{i-1}$ tab[j]) et i $\leq$ nbElem peuveut etre vrai apres chaque iterations aussi.

Cas d'induction: apres chaque iteration:

soit $tmp'_i$ et $i'$ des valeurs arbitraires de tmpi et i apres que la boucle a ete exécutée.

montrons que, $tmp'_i$=$tmp_i$+ tab[i]

et $i'$=i+1

est ce que $tmp'_i$ est valide?

$tmp'_i$=($\sum_{j=0}^{i}$ tab[j])= ($\sum_{j=0}^{i-1}$ tab[j])+tab[i]

donc, $tmp'_i$=$tmp_i$+tab[i] est vrai par induction.

## 3.3 Q3: validité

montrons la validité de somme(tab), on suppose maintenant que i $\leq$ nbElem est vrai et que la boucle se termine,

on a d'apres la supposition i $\leq$ n et quand on sort de la boucle on trouve que n$\leq$i donc on a n=i d'où,

$tmp_n$ = ($\sum_{j=0}^{n-1}$ tab[j]) est vrai sur tout l'algorithme.

# 4 Exo3: recherche sequentielle sur tab non trie

def Recherche(elem, tab):
i = 0
while elem!=tab[i] :
i = i + 1
return i

## 4.1 Q1:terminaison

Le corps de la boucle est composé d'instructions elementaires.

Nous sommes entrain de parcourir n elements d'un tableau donc il existe au plus n boucles de 1 operation.

## 4.2 Q2:invariant

Soit M(i): Si tab[j]!=elem , alors i!= $\sum_{j=0}^{i-1}$ j $\forall j \in \{0..i-1\}$ et tab[i]!=elem

Supposons que tab[j]!=elem
et Mq i!= $\sum_{j=0}^{i-1}$ j $\forall j \in \{0..i-1\}$ et tab[i]!=elem

Cas de base: Au debut de la fonction
on a , i=0 et donc le tableau est vide. vrai

Cas d'induction: En entrant dans la boucle mq i est toujours vrai. Soit i1 une valeur arbitraire de i telque i1=i+1 supposons que i= $\sum_{j=0}^{i-1}$ j $\forall j \in \{0..i-1\}$ est vrai (tout les elements qui precedent i ne sont pas egales a l'indice de elem).

on sait que tab[j]!=elem est vrai est que j=i-1
mq i1=i+1 est vrai
tab[i-1+1]!=elem donc tab[i]!=elem d'où la condition est vrai pour i1=i+1.

## 4.3 Q3: validité

Quand on sort de la boucle on a tab[i]=elem, et d'apres la demo par l'induction on a demontrer que tout les elem qui precedent i+1 ne sont pas egales a l'indice d elem donc i est le plus petit element

donc $\sum_{i=0}^{j-1}$ i est vrai sur tout l'algorithme

# 5 Exo5: pousser le plus grand elt d'un tab

def Push(tab,k): j=1
while(j¡k):
if tab[j]¡tab[j-1]:
tmp=tab[j-1]; tab[j-1]=tab[j]; tab[j]=tmp
j=j+1
print("j=",j," tab=",tab)

## 5.1 q1

executer push(tab,6)
pour le tab(0..7)=(8, 7, 12, 5, 15, 4, 3, 9) que fait push
push(tab,6)

en executant push(tab,6) la boucle se répéte 6 fois, et parcoure chaque couple des elts du tableau deux à deux En comparant chaque paire de couple d'elements et échange leurs emplacements s'il trouve que deux elements consecutifs ne sont pas trié d'une manière décroissante( c'est à dire que l'elt suivant d'un couple est inferieur a celui qui le precede du même couple) , jusqu'à le plus grand elements du tableau ce trouve dans le dernier emplacement du tab on peut dire aussi que le dernier elt du dernier couple est le plus grand elements du tableau.

pour tab(0..7) =(8, 7, 12, 5, 15, 4, 3, 9)
push(tab,2) = 7, 8, 12, 5, 15, 4, 3, 9
push(tab,3) = 7, 8, 12, 5, 15, 4, 3, 9 incg
push(tab,4) = 7, 8, 5, 12, 15, 4, 3, 9
push(tab,5) = 7, 8, 5, 12, 15, 4, 3, 9 inch
push(tab,6) = 7, 8, 5, 12, 4, 15, 3, 9

## 5.2 q2

mq push(tab,k) se termine

La boucle se repete ou avance k-1 fois car on est entrain de faire le parcour d'un tableau de k-1 elements et il y'a 4 operations.

donc push se termine.

## 5.3 q3

soit $tab_1$=tab pour j $\in$ (2...k), $tab_j$ designe le tab obtenus apres l'incrementation de j

L'invrariant de push: P(j): tab[j...n-1] inchangée $\forall j \in$ (1..k-1) et tab[0..j-1] contient son element max en position k-1

Par induction:
cas de base : pour j=1
tab1=tab donc tab[0] contient elt max et tab[1..n-1] reste inchangé vrai
cas d'induction: pour j=j+1 on a
Supp que P(j) vrai mq P(j+1) est vrai
$tab_j < tab_{j-1}$
donc P(i+1) est vrai par induction.

## 5.4 q4