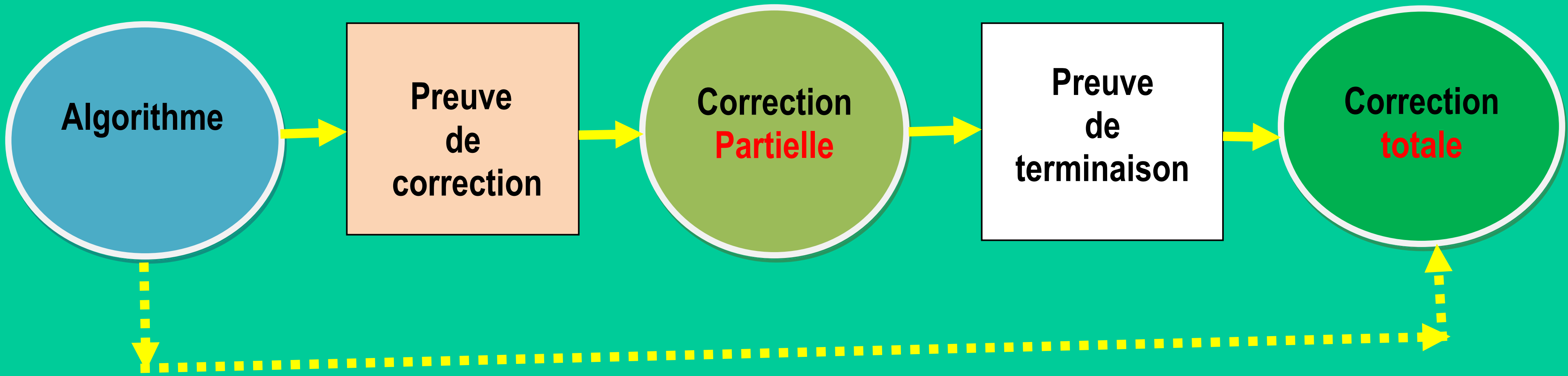
Td série n° 5 : sur la correction d'un algorithme

Un algorithme:

- qui est **partiellement correct**
- qui se termine **toujours**

est alors dit **totalement correct**.

En résumé


Algorithme
Preuve de correction
Correction Partielle
Preuve de terminaison
Correction totale

Comment apporter la preuve de correction ?

Preuve par induction

Preuve par assertion

I-La preuve par induction

En résumé, une **preuve par induction** est constituée de trois parties:

- une **hypothèse de récurrence**,
- une **base**
-et une **étape de récurrence**.

L'**hypothèse de récurrence** décrit l'énoncé à **prouver** : c'est le but de l'étape (3) dans l'exemple.

La **base** **prouve** le cas de départ : c'est l'étape (1) dans l'exemple.

L'**étape de récurrence** permet d'aller :

- de la **base**
- vers des **cas progressivement supérieurs**.

II- La preuve par assertions

- **Spécification**

$\{x \geqslant 0 \wedge y \geqslant 0 \wedge x = x_0 \wedge y = y_0\}$

```
while (y > 0)
    {
        int Save = y;
        y = x % y;
        x = Save;
    }
```

$\{x = pgcd(x_0, y_0)\}$

Précondition

Code à vérifier

Postcondition

Triplet de Hoare

Une portion du programme ou **code** est **correcte** si le **triplet de Hoare**:

{ P} **code** {Q}

est **vrai**.

P : **précondition**
Q : **post-condition**

Correction d'une boucle

On part du triplet de base suivant:

```
{P}
INIT
WHILE  C
        CORPS
FIN
  {Q}
```

pour construire le triplet:

$\{P\}$
INIT
$\{\ I\ \}$
WHILE C
$\{I \wedge C\}$
CORPS
$\{I\}$
$\{I \wedge \neg C\}$
FIN
$\{Q\}$

Pour prouver que le triplet est correct :

1-on met en évidence une assertion particulière **I**, appelée **invariant de boucle**.

L'**invariant** décrit une propriété **pendant** la boucle.

2-on doit prouver que successivement:

-**avant** la boucle :

$\{P\}$ **INIT** $\{ I \}$ est correct

-**pendant** la boucle

$\{I \wedge C\}$ **CORPS** $\{ I \}$ est correct

-**à la fin** de la boucle :

$\{I \wedge \neg C\}$ **FIN** $\{Q\}$ est correct

Si on a plusieurs boucles **imbriquées**, on les traite :

- **séparément**,
- en démarrant avec la boucle **la plus interne**.

Formalisation de la technique de preuve:

#Inv: invariant de la boucle while

#Inv : propriété vraie **avant** la boucle
while condition:

--montrer que si #Inv est vrai en **haut** de la boucle
 [**itération du while**]
--alors #Inv est vrai en **bas** de la boucle

finWhile
-- en déduire que #Inv est vrai **après** la boucle

Exercice 1 : calcul de la suite de Fibonacci

<u>Spécification:</u>

La suite de Fibonacci notée **(F_n)** est définie comme suit:

$F_0 = 0$;

$F_1 = 1$;

$F_2 = F_1 + F0$;

…

$F_n = F_{n-1} + F_{n-2}$

```
Fibonacci(n)
if n ≤ 1
        prev := n;
else
        { pprev := 0 ; prev := 1;
          i := 2
        }
while (i ≤ n)
        { f := prev + pprev;
          pprev := prev;    prev := f;
          i := i+1
        }
return (prev)
```

Mise en œuvre de la technique

```
Fibonacci(n)
-- {P}:   {n ≥ 0}
if n ≤ 1
        prev := n;
else
        {pprev := 0 ; prev := 1;
        i := 2
        }
        while (i ≤ n)
            {F := prev + pprev;    pprev := prev;     prev := F;
            i := i+1;
            }
-- {Q} : {prev = Fn}
return (prev)
```

Précondition

$\{P\} : \{n \geq 0\}$

Post-condition

$\{Q\}$: {Fibonacci(n) retourne prev = F_n}

Analyse de la condition

$\{n \geq 0 \text{ et } n \leq 1\}$

prev = n

$\{ prev = F_n \}$

Si n=0 alors **prev** =0

Comme $F_0 = 0$ alors {Fibonacci(0) retourne F_0}

Si n=1 alors **prev** =1

Comme $\mathbf{F_1 = 1}$ alors {Fibonacci(1) retourne F_1}

```
{ n > 1 }
pprev := 0      -- F0 ←0
prev := 1       -- F1 ←1
i := 2
while (i ≤ n)
        f := prev + pprev;      -- Fi ← Fi-1 + Fn-2
        pprev := prev;          -- Fi-2 ← Fi-1
        prev := f;              -- Fi-1 ← Fi-1 + Fi-2
        i := i + 1
{ prev = Fn }
```

$I = \{pprev = F_{i-2},\ \ prev = F_{i-1}\}$

Analyse de la boucle

Avant la boucle

$\{n > 1\}$

```
pprev := 0  ;
prev := 1;
i := 2;
```

$\{pprev = F_{i-2},\ prev = F_{i-1}\}$

correct

Dans la boucle

{pprev = F_{i-2} , prev = F_{i-1}, i $\leq$n} -- haut de la boucle
f := prev + pprev ;
pprev := prev;
prev := f;
i := i+1;
{pprev = F_{i-2} , prev = F_{i-1}} -- bas de la boucle

correct

Après la boucle

$\{pprev = F_{i\text{-}2}, \quad prev = F_{i\text{-}1}, \quad i = n+1\}$

$\{prev = F_n\}$

Correct

Etude de la terminaison

```
i = 2
while (i ≤ n)
        f := prev + pprev;
        pprev := prev;
        prev := f;
        i := i + 1;
```

Fonction de terminaison: F

$$F(i) = n - i + 1$$

- comme $i \leq n$ on a:

$$F(i) = n - i + 1 \mathbf{> 0}$$

- comme $i' = i + 1$ à chaque itération :

  $F(i)$ est **strictement décroissante**

Donc $F(i)$ finira par s'annuler :

$$F(i) = 0$$

$$\Rightarrow n - i + 1 = 0 \Rightarrow i = n+1$$

Cette condition la garantit la sortie du **<u>while</u>**, donc la **terminaison** de l'algorithme.

<u>Conclusion</u> :

L'algorithme est donc **correct** et se **termine : sa correction est totale.**

Exercice 2 : calcul de la factorielle

Soit l'algorithme suivant qui calcule **n !**

(1) lire (n)
(2) i := **2**
(3) fact := 1
(4) **<u>tant que</u>** i $\leq$ n **<u>faire</u>**
(5) fact := fact * i
(6) i := i + 1
<u>fintantque</u>
(7) écrire (fact)

Montrer la correction totale de l'algorithme précédent qui calcule n ! pour les entiers **n** $\geq$ **1**.

Pour établir la **correction totale**, il faut montrer que :

1- l'algorithme est **partiellement correct**,
2-la boucle **tant que** des lignes (4) à (6) doit terminer.

1-Comment prouver la correction partielle ?

Précondition : $n \geq 1$

Poscondition : fact = (i-1) !

invariant: $j=i \Rightarrow fact = (j-1)!$

<u>Pourquoi ?:</u>

A chaque fois qu'on atteint le test de la boucle :

$$i \leq n$$

avec la variable **i = j**, alors:

fact = (**j**-1) !

(1)lire (n) Précondition : $n \geq 1$

(2) i := 2

(3) fact := 1

#inv: $j=i \Rightarrow fact = (j-1)!$

(4) **tant que** $i \leq n$ **faire**

#inv1: $i=j \Rightarrow fact = (j-1)!$ **-- en "haut" de boucle**

(5) fact := fact * i

(6) i := i + 1

#inv2: $i=j \Rightarrow fact = (j-1)!$ **-- en "bas" de boucle**

fintantque Poscondition : fact = (i-1) !

(7) écrire (fact)

Avant la boucle

(1) lire (n)
(2) i := 2
(3) fact := 1
#inv: j=i ⇒ fact = (j-1)!

Comme on a:

$J = i = 2 \Rightarrow (j-1) = 1 = 1!$

fact =1

Il ressort immédiatement que:

fact = (j-1)!

En haut de la boucle

(4) **<u>tant que</u>** $i \leq n$ **<u>faire</u>**

#inv1: $i=j \Rightarrow fact = (j-1)!$

On suppose qu'en haut de boucle, l'invariant est vrai:

$$i=j \Rightarrow fact = (j-1)!$$

En bas de la boucle

#inv1: $i=j \Rightarrow fact = (j-1)!$
(5) fact := fact * i
(6) i := i + 1
#inv2: $i'=j' \Rightarrow fact' = (j'-1)!$
<u>fintantque</u>

On a:

$fact' = fact \times i$

$i' = i+1$

$j' = i' \Rightarrow fact' = fact \times i = (j-1)! \times j = j! = (j'-1)!$

<u>Conclusion</u> : l'algorithme est **partiellement correct**

Comment prouver la terminaison ?

Proposant la fonction de terminaison suivante :

$$F(i) = n-i+1$$

F est fonction **entière positive** car la **condition** d'itération de la boucle est :

$$\mathbf{i \leq n}$$

En effet :

$$i \leq n \Rightarrow \quad n\text{-}i \geq 0$$

$$\Rightarrow$$

$$\mathbf{F}(i) = n\text{-}i+1 > 0$$

Remarquons qu'à chaque itération :

- **i** := i+1
- **n** reste inchangé.

Ainsi **F** décroît de 1: **F**(i+1) = **F**(i) -1

Donc, $F(i)$ atteindra forcément la valeur **0**

Quand $F(i)$ devient nulle :

$$F(i) = n-i+1 = 0$$

On a :

$$i = n+1 \Rightarrow i > n$$

Ainsi, la condition de la boucle $\mathbf{i \leq n}$ sera fausse: la boucle **termine**.