Les invariants de boucles

Dans le cadre des programmes itératifs :

L'idée de départ : Il n'est pas toujours facile de :

- prédire avec certitude en sortie de boucle le contenu des variables et ainsi convaincre un interlocuteur que votre programme renvoie le bon résultat.
- proposer un algorithme itératif qui permet de résoudre un problème donné.

La notion d'invariant de boucle va nous aider à surmonter ces deux difficultés...

Définition : Un invariant de boucle est une propriété qui, malgré la modification des variables de boucles, reste vraie après chaque itération de la boucle.

Notation : Pour formuler un invariant de boucle faisant intervenir les variables a, b... on pourra noter a_k, b_k... les valeurs contenues dans les variables a, b... après la $k$ème itération de la boucle.

I] Exemples d'invariants de boucles

Exemple 1 : Factorielle

OCaml
```
let factoriel n = let res = ref 1 in
                  for k = 1 to n do res := !res * k
                                 done;
                  !res;;
```

On constate que la propriété P_k : $\ll res_k = k! \gg$ est toujours vraie.
C'est donc un invariant de boucle !

Exemple 2 : PGCD avec euclide

OCaml
```
let pgcd a b = let r0 = ref a and
                   r1 = ref b in
                   while !r1 <> 0 do let v = !r1 in
                                        r1 := !r0 mod !r1;
                                        r0 := v
                                     done;
                   !r0;;
```

P_k : $\ll r_{0,k} \wedge r_{1,k} = a \wedge b \gg$ est toujours vraie.
C'est donc un invariant de boucle !

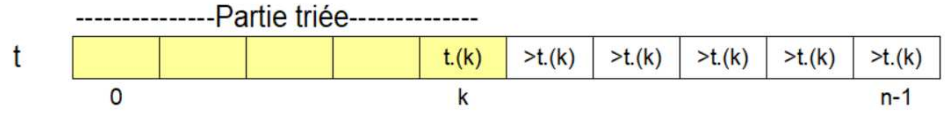
Exemple 3 : Tri par selection

OCaml
```
t : un tableau de longueur n

Pour k allant de 0 à (n-2) faire :
     placer le plus petit élément de t.(k...n-1) en position t.(k)
                          fin faire.
```

Q_k « `t.(0...k)` est trié » est toujours vraie.

P_k : « $\begin{cases} \texttt{t.(0...k)} \text{ est trié} \\ \text{les éléments de } \texttt{t.((k+1)...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...k)} \end{cases}$ » est toujours vraie.



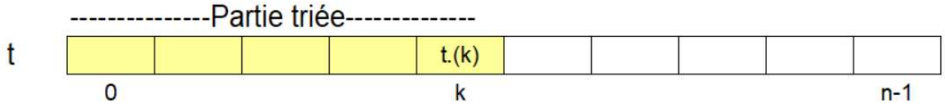
Ce sont donc des invariants de boucle, mais *seul le deuxième va nous permettre de prouver notre programme !*

Exemple 4 : Tri par insertion

OCaml
```
t : un tableau de longueur n

Pour k allant de 1 à (n-1) faire :
     placer t.(k) à sa place dans t.(0...k-1) en décallant les composantes d'un rang
                          fin faire.
```

P_k : « `t.(0...k)` est trié » est toujours vraie.



C'est donc un invariant de boucle !

Pour une boucle donnée, il n'y a pas "un" invariant de boucle, mais "des" invariants de boucle.

II] Preuve d'un programme

A retenir : Les instructions simples ne posent pas de difficulté.
En revanche, les traitements effectués dans les boucles peuvent faire l'objet de doutes.
Pour prouver le résultat d'une boucle, on utilise un invariant de boucle **bien choisi**.

Reprenons les exemples précédents...

La fonction factorielle :

- invariant de boucle : P_k « $res_k = k!$ »
- Sortie de boucle : On a $k = n$ et donc « $res_n = n!$ »

La variable `res` contient bien la valeur $n!$.

La fonction PGCD :

- invariant de boucle : P_k : « $r_{0,k} \wedge r_{1,k} = a \wedge b$ »
- Sortie de boucle : On a $r_{1,k} = 0$ et donc $r_{0,k} \wedge 0 = a \wedge b$, c'est à dire $r_{0,k} = a \wedge b$ »

La variable `r0` contient bien la valeur $a \wedge b$.

Tri par selection :

- invariant de boucle : P_k : « $\begin{cases} \texttt{t.(0...k)} \text{ est trié} \\ \text{les éléments de } \texttt{t.((k+1)...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...k)} \end{cases}$ »
- Sortie de boucle : On a $k = n-2$ et donc « $\begin{cases} \texttt{t.(0...(n-2))} \text{ est trié} \\ \texttt{t.(n-1)} \text{ est plus grand que les éléments de } \texttt{t.(0...(n-2))} \end{cases}$ »

Par conséquent en fin de boucle, la tableau `t` est bien trié.

Question : Et si nous avons pris simplement P_k « `t.(0...k)` est trié » comme invariant de boucle ?

Tri par insertion :

- invariant de boucle : P_k « `t.(0...k)` est trié »
- Sortie de boucle : On a $k = n - 1$ et donc « `t.(0...(n-1))` est trié »

En fin de boucle, le tableau `t` est bien trié !

Conclusion : Le choix judicieux d'un invariant de boucle nous permet de prouver très facilement le résultat en sortie d'une boucle.

III] Comment savoir si une propriété est bien un invariant de boucle ?

A retenir : Soit P_k : « On a ... après l'itération k » une propriété.
Pour affirmer que P_k est un invariant de boucle, il faut le prouver ! !

Méthode pour prouver que P_k est un invariant de boucle :

On procède tout simplement par récurrence.

1. On montre que la propriété P_k est vraie avant (ou parfois "après") la première itération.
2. On suppose que P_k est vraie et on montre alors que P_{k+1} est également vraie.
3. On peut alors conclure.

Exemple 1 : Factorielle

OCaml
```
let factoriel n = let res = ref 1 in
                  for k = 1 to n do res := !res * k
                                 done;
                  !res;;
```

Montrons que P_k : $\ll res_k = k! \gg$ est un invariant de boucle.

- Avant la boucle : $\begin{cases} res_0 = 1 \\ 0! = 1 \end{cases}$ par conséquent, on a bien $P_0 : res_0 = 0!$
- Hérédité : Supposons que $res_k = k!$ et montrons que $res_{k+1} = (k+1)!$.

  L'instruction `res := !res * k` nous donne la relation $res_{k+1} = res_k * (k+1)$ et donc : $res_{k+1} = (k+1)!$

Exemple 2 : PGCD avec euclide

OCaml
```
let pgcd a b = let r0 = ref a and
                   r1 = ref b in
                   while !r1 <> 0 do let v = !r1 in
                                     r1 := !r0 mod !r1;
                                     r0 := v
                                  done;
                   !r0;;
```

Montrons que P_k : $\ll r_{0,k} \wedge r_{1,k} = a \wedge b \gg$ est un invariant de boucle.

- Avant la boucle : On a $\begin{cases} r_0 = a \\ r_1 = b \end{cases}$ et par conséquent, on a bien $r_0 \wedge r_1 = a \wedge b$.
- Hérédité : Supposons que $r_{0,k} \wedge r_{1,k} = a \wedge b$ et montrons que $r_{0,k+1} \wedge r_{1,k+1} = a \wedge b$.

  Les instructions nous disent que :

  → $r_{1,k+1}$ est le reste de la division euclidienne de $r_{0,k}$ par $r_{1,k}$
  Notre cours d'arithmétique nous dit alors que $r_{0,k} \wedge r_{1,k} = r_{1,k} \wedge r_{1,k+1}$

  → $r_{0,k+1} = r_{1,k}$ et nous avons donc $r_{0,k} \wedge r_{1,k} = r_{0,k+1} \wedge r_{1,k+1}$

  Finalement, d'après l'hypthèse de récurrence : $r_{0,k+1} \wedge r_{1,k+1} = a \wedge b$

Exemple 3 : Tri par sélection

OCaml
```
t : un tableau de longueur n

Pour k allant de 0 à (n-2) faire :
     placer le plus petit élément de t.(k...n-1) en position t.(k)
                             fin faire.
```

Soit P_k : « $\begin{cases} \texttt{t.(0...k)} \text{ est trié} \\ \text{les éléments de } \texttt{t.((k+1)...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...k)} \end{cases}$ »

Montrons que P_k est un invariant de boucle.

- Après la première itération :

  La première itération consiste à placer le plus petit élément de `t.(0...n-1)` en position `t.(0)`.

  Par conséquent, nous avons bien : $\begin{cases} \texttt{t.(0...0)} \text{ est trié} \\ \text{les éléments de } \texttt{t.(1...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...0)} \end{cases}$

- Hérédité : Supposons que $\begin{cases} \texttt{t.(0...k)} \text{ est trié} \\ \text{les éléments de } \texttt{t.((k+1)...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...k)} \end{cases}$.

  Montrons que $\begin{cases} \texttt{t.(0...(k+1))} \text{ est trié} \\ \text{les éléments de } \texttt{t.((k+2)...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...(k+1))} \end{cases}$.

  Pour la valeur `k+1`, l'itération consiste à

  ```
  << placer le plus petit élément de t.((k+1)...n-1) en position t.(k+1) >>
  ```

  Or $\begin{cases} \texttt{t.(0...k)} \text{ est trié} \\ \text{les éléments de } \texttt{t.((k+1)...(n-1))} \text{ sont plus grands que ceux de } \texttt{t.(0...k)} \end{cases}$.

  Après cette itération, nous avons alors :

  → le tableau `t.(0...(k+1))` est trié
  → les éléments de `t.((k+2)...(n-1))` sont plus grands que ceux de `t.(0...(k+1))`

Exemple 4 : Tri par insertion

OCaml
```
t : un tableau de longueur n

Pour k allant de 1 à (n-1) faire :
     placer t.(k) à sa place dans t.(0...k-1) en décallant les composantes d'un rang
                            fin faire.
```

Montrons que P_k : « `t.(0...k)` est trié » est un invariant de boucle.

- Avant la première itération : la tableau `t.(0...0)` est bien trié.

- Hérédité : Supposons que `t.(0...k)` est trié et montrons que `t.(0...(k+1))` est trié.
  A l'étape `k+1`, l'instruction « `placer t.(k) à sa place dans t.(0...k-1) en décalant les composantes d'un rang` » nous donne directement le fait que `t.(0...(k+1))` est trié.

Exemple 5 : Algorithme de Bezout

Rappel de la méthode : On souhaite déterminer deux entiers u et v tels que $au + bv = a \wedge b$.
Pour cela, on programme à l'aide d'une boucle `while` les 4 suites suivantes :

- La suite (r_k) définie par $\begin{cases} r_0 = a \\ r_1 = b \end{cases}$ et r_{k+2} est le reste de la DE de r_k par r_{k+1}.

  Il s'agit de la suite des restes d'euclide... Le dernier reste non nul donnant alors $a \wedge b$.

- La suite (q_k) des quotients de la DE de r_{k-2} par r_{k-1}.
- Les suites (u_k) et (v_k) définies par $\begin{cases} u_0 = 1 \\ u_1 = 0 \end{cases}$ et $\begin{cases} v_0 = 0 \\ v1 = 1 \end{cases}$ et les relations de récurrences :

$$\begin{cases} u_{k+2} = u_k - q_{k+1}u_{k+1} \\ v_{k+2} = v_k - q_{k+1}v_{k+1} \end{cases}$$

1. Proposer une implémentation OCaml de cet algorithme.
2. Prouver que P_k : « $au_k + bv_k = r_k$ » est un invariant de boucle.
3. En déduire la validité de l'algorithme.

Réponse : Voir sur le forum...

IV] Comment savoir quel invariant de boucle choisir ?

A retenir :

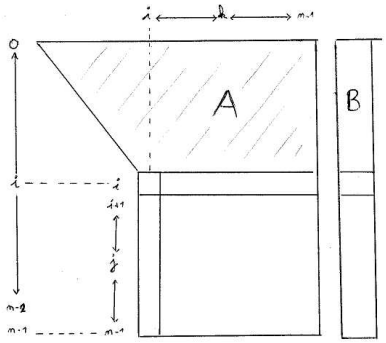
L'invariant de boucle le plus simple est, comme dans l'exemple de la fonction `factoriel`, celui qui prédit le contenu de chacune des variables de la boucle. Cependant définir un tel invariant de boucle n'est pas toujours possible en pratique. Pensez à la fonction `pgcd` précédente.

Dans la plupart des cas, le choix de l'invariant de boucle est alors intuitif... On peut cependant évaluer a priori sa pertinence en se demandant si, en l'appliquant en sortie de boucle, il peut nous permettre de prouver notre programme.

Exemples d'invariants de boucles intuitifs

1. Calcul de la somme des n premiers entiers : « $s_k = \frac{k(k+1)}{2}$ »
2. Triangularisation d'un système avec Gauss :

   « Le système obtenu est équivalent au système initial et la matrice a la forme suivante : »



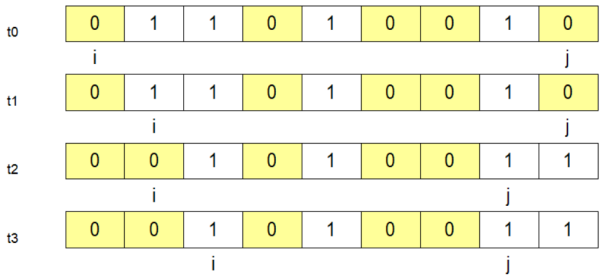
Pour choisir un invariant de boucle on peut appliquer sur un exemple les premières itérations d'une boucle.

EXO 1 : Un algorithme de tri

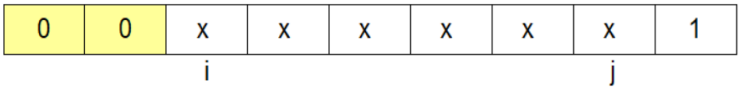
Prouver que le programme suivant permet de trier un tableau de 0 et de 1.

OCaml
```
let tri t = let n = Array.length t in
            let i = ref 0 and
                j= ref (n-1) in
            while i < j do match t.(!i) with
                           | 0 -> i := !i + 1
                           | 1 -> let v = t.(!i) in echange t.(!i) t.(!j);
                                                    j := !j - 1
                        done;;
```

Réponse : Appliquons l'algorithme au tableau `t = [|0;1;1;0;1;0;0;1;0|]` .

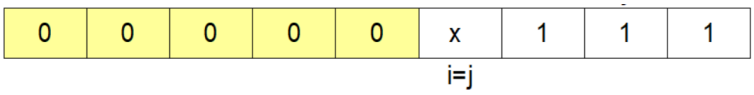


Il semblerait que l'invariant de boucle soit :



Preuve du programme :

- On commence par vérifier que l'on a bien un invariant de boucle... A faire sur le forum...
- Sortie de boucle :
  → En sortie de boucle, nous avons $i = j$
  → L'invariant de boucle nous permet d'affirmer que nous avons alors :



Que `x` vaille 0 ou 1, le tableau obtenu en fin de boucle est bien trié.

Lorsque l'invariant de boucle n'est pas facile à déterminer, il peut être donné par l'énoncé.

EXO 2 : Fonction mystère

OCaml
```
let mystere a n = let b = ref a and
                      m = ref n and
                      res = ref 1 in
                  while !m > 0 do if !m mod 2 = 1 then res := !res * !b;
                                  b := !b * !b;
                                  m := !m / 2
                                done;
                  !res;;
```

1. Montrer que $\ll res * b^m = a^n \gg$ est un invariant de boucle.
2. En déduire ce que renvoie la fonction `mystere`

Réponse :

1. Montrons que $\ll res * b^m = a^n \gg$ est un invariant de boucle :

   - Avant la première itération, nous avons $\begin{cases} res = 1 \\ b = a \\ m = n \end{cases}$ et donc on a bien : $res * b^m = a^n$.
   - On suppose avoir $res * b^m = a^n$ après la kième itération. $(res_k * b_k^{m_k} = a^n)$

     Après la (k+1)ième, les instructions de boucle nous donne :

     $\rightarrow$ Si $m_k = 2p + 1$ alors $\begin{cases} res_{k+1} = res_k * b_k \\ b_{k+1} = b_k * b_k \\ m_{k+1} = p \end{cases}$ et donc

     $$res_{k+1} * b_{k+1}^{m_{k+1}} = res_k * b_k * (b_k * b_k)^p = res_k * b_k^{2p+1} = res_k * b_k^{m_k} = a^n$$

     $\rightarrow$ Si $m_k = 2p$, alors ... (à faire vous même...)

2. Résultat renvoyé par `mystere a n` :

   En sortie de boucle, nous avons $m_k = 0$ avec l'invariant de boucle $res * b^m = a^n$, ce qui nous donne :

   $$res_k = a^n$$

   La variable `res` contient donc la valeur a^n.
   `mystere a n` renvoie donc a^n.

EXO 3 : Division euclidienne

Soient $a,\ b \in \mathbb{N}^*$.
Prouver que le programme suivant renvoie bien le quotient et le reste de la division euclidienne de a par b.

Vous utiliserez l'invariant de boucle suivant : $\begin{cases} a = bq + r \\ r \in \mathbb{N} \\ q \in \mathbb{N} \end{cases}$.

OCaml
```
let div_eucl a b = let q = ref 0 and
                       r = ref a in
                   while !r >= b do q := !q + 1;
                                    r := !r - b
                                 done;
                   q,r;;
```

Réponse :

1. Preuve de l'invariant de boucle :

   (a) Avant la première itération, nous avons bien : $bq + r = b * 0 + a = a$ avec $\begin{cases} r = a \in \mathbb{N} \\ q = 0 \in \mathbb{N} \end{cases}$.

(b) Supposons avoir $\begin{cases} a = bq_k + r_k \\ r_k \in \mathbb{N} \\ q_k \in \mathbb{N} \end{cases}$ après la kème itération.

Supposons que la (k+1)ème itération a lieu (cad $r_k \geq b$).

Compte-tenu des instructions $\begin{cases} \texttt{q := !q + 1} \\ \texttt{r := !r - b} \end{cases}$ et de l'hypothèse de récurrence, nous avons :

$$bq_{k+1} + r_{k+1} = b(q_k + 1) + r_k - b = bq_k + r_k = a$$

Comme nous avons $\begin{cases} r_k \in \mathbb{N} \\ r_k \geq b \end{cases}$ et $q_k \in \mathbb{N}$, alors on a également $\begin{cases} r_{k+1} \in \mathbb{N} \\ q_{k+1} \in \mathbb{N} \end{cases}$.

Ce qui prouve notre invariant de boucle.

2. Sortie de boucle : En sortie de boucle, nous avons $r \in [\![0, b-1]\!]$, $q \in \mathbb{N}$ et $a = bq + r$.

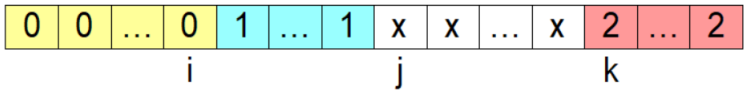
Par conséquent, $\begin{cases} q \\ r \end{cases}$ sont respectivement les $\begin{cases} \text{quotient} \\ \text{reste} \end{cases}$ de la division euclidienne de a par b.

V] Concevoir un algorithme à partir d'un invariant de boucle

L'idée : On peut envisager la conception d'un algorithme à partir de la donnée d'un invariant de boucle.

EXO 1 : Tri du drapeau

On souhaite trié un tableau ne contenant que les valeurs 0, 1 et 2.
Proposer un algorithme qui vérifie l'invariant de boucle suivant :



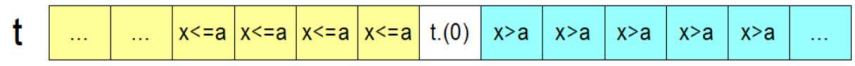
Réponse :

- A chaque itération, on examine l'élément `t.(j)`
- On a alors les différentes situations suivantes :
  - → Si `t.(j) = 0` alors on fait `t.(i+1) <-> t.(j)` puis `i = i + 1` et `j = j + 1`
  - → Si `t.(j) = 1` alors on fait `j = j + 1`
  - → Si `t.(j) = 2` alors on fait `t.(k-1) <-> t.(j)` puis `k = k - 1`
- On effectue ce travail tant que `j <> k`.
- On initialise `i`, `j` et `k` en prenant : $\begin{cases} \texttt{i = -1} \\ \texttt{j = 0} \\ \texttt{k = n} \end{cases}$

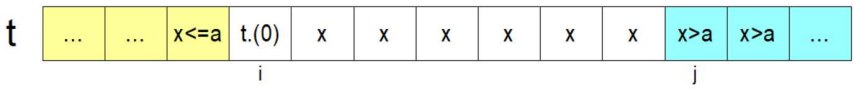
Vous pouvez maintenant vous entrainer à donner une implémentation OCaml de cet algorithme.

EXO 2 : Découpage d'un tableau

On note `t.(0) = a`, la première composante initiale d'un tableau `t` de taille n.
On souhaite répartir "en place" les éléments de `t` de façon que :



Proposer un algorithme qui vérifie l'invariant de boucle suivant :



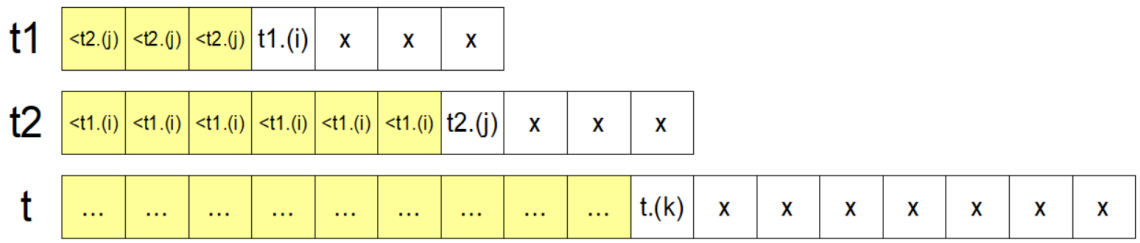
Réponse :

- On note chaque itération, on examine l'élément `t.(i+1)`
- On a alors les différentes situations suivantes :
  → Si `t.(i+1) <= t.(0)` alors on fait `t.(i+1) <-> t.(i)` puis `i = i + 1`
  → Si `t.(i+1) > t.(0)` alors on fait `t.(j-1) <-> t.(i)` puis `j = j - 1`
- On effectue ce travail tant que `i <> j-1`.
- On initialise les références `i` et `j` à $\begin{cases} \texttt{i = 0} \\ \texttt{j = n} \end{cases}$

Vous pouvez maintenant vous entrainer à donner une implémentation OCaml de cet algorithme.

EXO 3 : Fusion des 2 tableaux triés

On dispose de deux tableaux d'entiers `t1` et `t2` que l'on souhaite fusionner en un seule tableau trié `t`.
Proposer un algorithme qui vérifie l'invariant de boucle suivant :



Les éléments en jaune ayant été correctement placés dans le tableau `t`.

Réponse :

- On note chaque itération, on compare les éléments `t1.(i)` et `t2.(j)`
- On a alors les différentes situations suivantes :
  → Si `t1.(i) <= t2.(j)` alors on fait `t.(k) <- t1.(i)` puis `i = i + 1` et `k = k + 1`
  → Si `t1.(i) > t2.(j)` alors on fait `t.(k) <- t2.(j)` puis `j = j + 1` et `k = k + 1`

- On initialise `i = j = 0`.
- On effectue ce travail tant que `i < n1` et `j < n2`.
  Si `k <> n1 + n2` il faut finir de compléter le tableau avec les éléments du tableau qui n'a pas été épuisé.

Vous pouvez maintenant vous entrainer à donner une implémentation OCaml de cet algorithme.

A retenir : ♡ Lorsque vous devez proposer un algorithme de résolution d'un problème, pensez à préciser l'invariant de boucle sur lequel repose votre algorithme. Cela permettra au correcteur de beaucoup mieux comprendre vos explications. ♡

VI] Exercices

Exercice : 1

On souhaite construire une fonction itérative qui reconnaît si une chaine de caractères est un palindrome ou pas.

1. Proposer un invariant de boucle judicieux.
2. En déduire un algorithme de résolution du problème.
3. En déduire une fonction OCaml répondant à la question.

Réponse :

On remarque que la propriété suivante est un invariant de boucle :

Exercice : 2

(∗) Que renvoie le programme suivant appliqué à $p = [|a_0;\ a_1;\ \ldots; a_n|]$ et $x \in \mathbb{R}$?

OCaml
```
let f p x = let s = ref 0 and
                n = (Array.length p -1) in
            for k = 0 to n do s := x * !s + p.(n-k)
                              done ;
            !s ;;
```

Réponse :

On remarque que $P_k : s_k = a_n x^k + a_{n-1}x^{k-1} + \cdots + a_{n-k}$ est un invariant de boucle.

Exercice : 3

(∗) Le programme suivant permet de calculer la valeur de $\sqrt{x}$ à ε près en calculant les termes u_n de la suite :

$$(u_n) \text{ définie par } \begin{cases} u_0 = 1 \\ u_{n+1} = \dfrac{1}{2}(u_n + \dfrac{x}{u_n}) \end{cases}$$

OCaml
```
let sqrt2 x eps = let u0 = ref 1. in
              let n = log((1.+.x)/.(2.*.eps))/.log(2.) +. 1. in
              for k = 1 to int_of_float(n)+1 do u0 := (!u0 +. x/.(!u0))/.2.
                                            done ;
              !u0 ;;

sqrt2 2. 0.0001;;
sqrt 2.;;

# sqrt2 2. 0.0001;;
- : float = 1.4142135623730949
# sqrt 2.;;
- : float = 1.4142135623730951
```

1. Prouver sa validité en admettant que "$0 \leq u_n - \sqrt{x} \leq \frac{u_1}{2^{n-1}}$" est un invariant de boucle.
2. Prouver que la propriété suivante est bien un invariant de boucle.

   Aide : On pourra remarquer que pour tout $n \geq 1$, on a $\sqrt{x} \geq \frac{x}{u_n}$ et $u_{n+1} - \sqrt{x} = \frac{1}{2}((u_n - \sqrt{x}) - (\sqrt{x} - \frac{x}{u_n}))$.

Exercice : 4

On propose l'algorithme suivant portant sur un tableau `t` de longueur n indexé de 0 à $n-1$:

```
i := 0
j := n-1
Tant que i<j faire : si t(i)<=t(j) alors i := i+1
                                   sinon j := j-1
Renvoyer t(i)
```

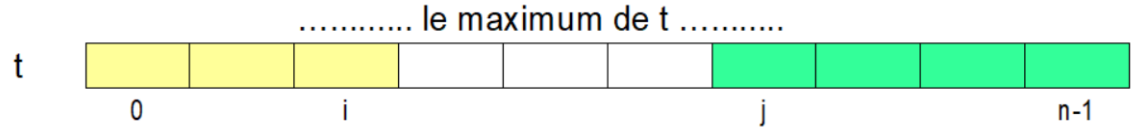
1. Conjecturer le résultat renvoyé par ce programme.
2. Prouver votre conjecture en proposant un invariant de boucle judicieux.

Réponse :

1. En testant cette fonction sur des exemples, il semblerait que le programme renvoie le maximum des composantes.

2. Pour le prouver , on considère l'invariant de boucle suivant :

   La maximum de `t` a un indice compris entre `i` et `j`



Exercice : 5

Le crible d'érathostène permet de déterminer les premiers nombres premiers inférieurs à un entier n en procédant de la façon suivante.

- On sélectionne 2, puis on "raye" les multiples de 2 compris entre 3 et n.
- On sélectionne l'entier suivant non rayé et on élimine ses multiples
- On répète l'opération précédente tant que l'entier sélectionné est inférieur ou égal à $\sqrt{n}$.
- Les éléments restants sont alors les nombres premiers cherchés.

1. Proposer et démontrer un invariant de boucle pour cet algorithme.

2. En déduire que cet algorithme renvoie bien le résultat attendu.
3. Proposer une implémentation OCaml.

On pourra considérer un tableau initial `t = [1;1;...;1|]` *de longueur* $n+1$ *et on "rayera le nombre* k *non premier" en annulant la composante* `t.(k)` *" :* `t.(k) <- 0`*.*