a) Ecrivez un programme en pseudo-langage qui effectue la permutation circulaire d'un vecteur a de taille 5, de la gauche vers la droite.

Votre programme pourra comporter le preambule :

```
// initialisation de a
a <- [4,2,8,6,9]
```

Et pour afficher un vecteur vous pourrez ecrire :

```
// affichage du resultat
affiche a
```

A titre d'exemple le resultat de l'execution du programme sur $[4, 2, 8, 6, 9]$ sera $[9, 4, 2, 8, 6]$

b) Parmis ces propositions, lesquelles sont des invariants du programme suivant :

```
....
tableau a        : un tableau d'entiers de dimention dim
s,i              : deux entiers

debut test
   // initialisation
   init a
   s <- 0
   // calcul bizzare
   pour i <- 1 a dim faire
        s <- s + a[i] + i-1
   fin pour
fin test
```

$a)\ s = \Sigma_{j=1}^{i}(a[j] + j)$	$b)\ s = \Sigma_{j=1}^{i-1}(a[j] + j - 1)$
$c)\ s = \Sigma_{j=1}^{i}(a[j] + j - 1)$	$d)\ aucun$

```
Programme Permutation

      Constantes
         dim = 5

      Types
         vecteur = tableau de dim entiers

      Variables
         a    : vecteur
         i    : un entier pour enumerer les cases
         sauv : un entier pour se rappeler du contenu de la derniere case

      Debut Permutation

         // initialisation de a
         a <- [4,2,8,6,9]

         // affichage du vecteur de depart
         affiche `` la permutation de'' a ``est :'' endl

         //
         // corps de la permutation
         //

         // 1) on se rapelle du contenu de la derniere case
         sauv <- a[dim]

         // 2) on effectue la permutation de toutes les cases vers la droite,
         // sauf pour la derniere
         // pour ne rien effacer, on commence par la droite
         pour i <- dim a 2 faire
              a[i] <- a[i-1]
         fin pour

         // reste a permuter la derniere
         a[1] <- sauv

        //
        // affichage du resultat
        //
        affiche a

fin Permutation
```

a) On considère un programme contenant un vecteur a de dimention 10. On demande à l'utilisateur d'entrer une position p dans ce tableau, afin de casser ce vecteur entre la position p et $p + 1$, puis d'inverser les deux morceaux en le recopiant dans un vecteur b.

Exemple : si $a = [5, 3, 6, 2, 8, 0, 4, 7, 8, 6]$ et que p vaut 3, il faut donc inverser les blocs $[5, 3, 6]$ et $[2, 8, 0, 4, 7, 8, 6]$, pour obtenir $b = [2, 8, 0, 4, 7, 8, 6, 5, 3, 6]$.

Vous devez ecrire ce programme en pseudo-langage. Il contiendra en particulier la partie de code suivante :

```
            ....
// initialisation de a
init a
affiche a
// l'utilisateur precise ou a lieu la cassure
afficher ``quelle est la position ou je dois casser le vecteur ?''
lire p
```

```
//
// votre partie du programme pour l'inversion
//
             ....
//
// affichage du resultat
//
affiche b
             ....
```

b) Parmis ces propositions, lesquelles sont des invariants du programme suivant :

```
....
tableau a        : un tableau d'entiers de dimention dim
p,i              : deux entiers

debut test
   // initialisation
   init a
   p <- 1
   // calcul bizzare
   pour i <- 1 a dim faire
        p <- p * a[dim-i+1]
   fin pour
fin test
```

a) $p = \Pi_{j=1}^{dim}(a[j])$	*b)* $p = \Pi_{j=1}^{i-1}(a[dim-j+1])$
c) $p = \Pi_{j=1}^{i}(a[dim-j+1])$	*d)* $aucun$

```
Programme Cassure

      Constantes
         dim = 10

      Types
         vecteur = tableau de dim entiers

      Variables
         a,b   : vecteur
         i     : un entier pour enumerer les cases
         p     : un entier pour la cassure

      Debut Cassure
        // initialisation de a
        init a
        affiche a

        // l'utilisateur precise ou a lieu la cassure
        afficher ``quelle est la position ou je dois casser le vecteur ?''
        lire p

        //
        // votre partie du programme pour l'inversion
        //

        // le bloc gauche de a passe a droite de b
        pour i <- 1 a p faire
             b[dim-p +i] <- a[i]
        fin pour

        // le bloc droit de a passe a gauche de b
        pour i de p+1 a dim faire
             b[i-p] <- a[i]
        fin pour

        //
        // affichage du resultat
        //
        affiche b

fin Cassure
```