Énoncé

Dans tout l'exercice, `n`, `p`, `A` sont des variables (*globales* si cette notion a été traitée) qui désignent respectivement deux entiers naturels non nuls et un tableau indexé de 1 à la valeur de `n`.

On modélise une fonction f de $\{1, 2, \cdots, n\}$ vers $\{1, 2, \cdots, p\}$ à l'aide du tableau `A` en associant , pour tout i entre 1 et n, la valeur $f(i)$ à la clé i.

Vous devez former le diagramme et écrire le code Maple qui permet d'exécuter les tâches précisées. Si la syntaxe des procédures a été vue en TD, définir les procédures dont le nom est indiqué au début de la question. Tester sur des exemples de votre choix.

1. `est_image(x)`. Renvoyer vrai si x est une image de la fonction f et faux sinon.
2. `nbantecedent(x)` Renvoyer le nombre d'antécédents de x pour la fonction f lorsque x est une image et renvoyer 0 si x n'est pas une image.
3. `est_surjective()` Renvoyer vrai si f est surjective et faux sinon.
4. `est_injective()` Renvoyer vrai si f est injective et faux sinon.
5. `reciproque()` Renvoyer un tableau `B` qui, lorsque f est bijective, modélise la bijection réciproque f^{-1}.

Corrigé

Fichiers Maple à télécharger : Csurjinc.mw ou Csurjinc.mws

1. `est_image(x)`. Renvoyer vrai si x est une image de la fonction f et faux sinon.
   Syntaxe sans utiliser de fonction.

```
> n:=4;p:=5;
  A:= array(1..n,[4,4,2,1]);
  x:= 3;

  ## x est-il une valeur du tableau ?
  res := false:
  for i from 1 to n do
    if A[i]=x then
      res := true;
      i := n+1; # pour sortir de la boucle
    fi;
  od:
  res;
```

   Syntaxe avec une fonction

```
> # définition de la fonction
  est_image := proc(x)
    local i, res;
    global n,p,A;
    res := false:
    for i from 1 to n do
      if A[i]=x then
        res := true;
        i := n+1; # pour sortir de la boucle
      fi;
    od:
    return(res);
  end proc:

  # définition des données
  n:=4;p:=5;
  A:= array(1..n,[4,4,2,1]);
```

```
# appel de la fonction
est_image(3);
```

2. `nbantecedent(x)` Renvoyer le nombre d'antécédents de x pour la fonction f lorsque x est une image et renvoyer 0 si x n'est pas une image.
   Syntaxe sans fonction

```
> n:=4;p:=5;
  A:= array(1..n,[4,4,2,1]);
  x:= 4;

  ## Nombre d'antécédents de x?
  res := 0:
  for i from 1 to n do
    if A[i]=x then
      res := res +1;
    fi;
  od:
  res;
```

   Syntaxe avec une fonction

```
> #définition de la fonction
  nbantecedent := proc(x)
    local i, res;
    global n,p,A;
    res := 0:
    for i from 1 to n do
      if A[i]=x then
        res := res +1;
      fi;
    od:
    return(res);
  end proc:

  #définition des données
  n:=4;p:=5;
  A:= array(1..n,[4,4,2,1]);

  #appel de la fonction
  nbantecedent(4);
```

3. `est_surjective()` Renvoyer vrai si f est surjective et faux sinon.
   Syntaxe sans fonction. On initialise la réponse à `true` puis on boucle sur les éléments de l'espace d'arrivée. On s'arrête en renvoyant `false` lorsqu'un élément n'est pas une image.

```
> n:=5;p:=4;
  A:= array(1..n,[4,4,2,1,3]);

  # la fonction est-elle surjective ?
  res:= true:
  for x from 1 to p do
    est_pas_im := true;
    for i from 1 to n do
      if A[i] = x then
        est_pas_im:= false;
        i:= n+1;
      fi;
    od;
    if est_pas_im then
      res := false;
      x := p+1;
    fi;
  od:
  res;
```

   Avec des fonctions, l'idée est la même mais on peut réutiliser `est_image`.

```
> # définition de la fonction
  est_surjective := proc()
    local x, res;
    global n,p,A;
    x:= 1;
    while est_image(x) and x<=p do
      x := x+1;
    od;
    res := evalb(x=p+1);# évalue en booléen
    return(res)
  end proc:

  # définition des données
  n:=5;p:=5;
```

```
A:= array(1..n,[4,4,2,1,3]);

#appel de la fonction
est_surjective();
```

4. `est_injective()` Renvoyer vrai si f est injective et faux sinon.
   L'idée est de chercher un couple d'éléments distincts dans l'ensemble de départ qui ont la même image. Si on en trouve un, la fonction n'est pas injective. Comme les éléments sont distincts, on en prend un strictement plus petit que l'autre.

```
> n:=5;p:=7;
  A:= array(1..n,[4,6,2,1,3]);

  # la fonction est-elle injective ?
  res:= true:
  for i from 1 to n do
    for j from 1 to i-1 do
      if A[i] = A[j] then
        res := false;
        j:= n+1;
        i:= n+1; # pour sortir des boucles
      fi;
    od:
  od:
  res;
```

```
> # définition de la fonction
  est_injective := proc()
    local i,j,res;
    global n,p,A;
    res:= true:
    for i from 1 to n do
      for j from 1 to i-1 do
        if A[i] = A[j] then
          res := false;
          j:= n+1;
          i:= n+1; # pour sortir des boucles
        fi;
      od:
    od:
    return(res);
  end proc:

  # définition des données
  n:=5;p:=7;
  A:= array(1..n,[4,6,2,1,3]);

  # appel
  est_injective();
```

5. `reciproque()` Renvoyer un tableau `B` qui, lorsque f est bijective, modélise la bijection réciproque f^{-1}.
   On suppose ici que la fonction est bijective. On ne le vérfiera donc pas. On remplit les *clés* du tableau `B` avec les valeurs du tableau `A`.

```
> n:=5;p:=5;
  A:= array(1..n,[4,5,2,1,3]);

  # former un tableau "réciproque"
  B:= array(1..p):
  for i from 1 to n do
    B[A[i]] := i;
  od:
  eval(B);
```

   Syntaxe avec une fonction :

```
  # définition de la fonction
  reciproque := proc()
    local i,B;
    global n,p,A;
    B:= array(1..p):
    for i from 1 to n do
      B[A[i]] := i;
    od:
    return(eval(B));
  end proc:

  # définition des données
  n:=5;p:=5;
  A:= array(1..n,[4,5,2,1,3]);
```

```
# appel
C := reciproque():
eval(C);
```

Noter l'utilisation du `eval()` avant le renvoi pour forcer l'évaluation. En effet les tableaux dérogent à la règle de l'évaluation maximale. Quand une variable désigne un tableau, elle est évaluée à son nom seul. Il me semble préférable d'éviter le renvoi d'un nom local à la fonction.