Solution && barème

I- Construire un programme avec vérification

A- Nous souhaitons écrire un programme qui reçoit un tableau X de N entiers et un tableau Y de deux entiers. Ce programme retourne dans les variables p et d les indices de la première position (p) et de la dernière position (d) de Y dans X :

p vérifie : $(Y[0] = X[p] \wedge Y[1] = X[p+1])$ /* noté dans les commentaires (Y[0..1] = X[p..p+1])*/
d vérifie : (Y[0..1] = X[d..d+1])

On sait que le tableau Y[0..1] est présent au moins une fois dans le tableau X.

Spécification :

/* $(N>1) \wedge (\exists I : 0 \leq I \leq N-2 \wedge (Y[0] = X[I] \wedge Y[1] = X[I+1]))$ */

Calculer (X, Y, N, p, d)

/* $(0 \leq p \leq d \leq N-2) \wedge (Y[0..1] = X[p..p+1]) \wedge (Y[0..1] = X[d..d+1])$
$\wedge (\forall I : 0 \leq I < p \rightarrow X[I..I+1] \# Y[0..1]) \wedge (\forall I : d < I \leq N-2 \rightarrow X[I..I+1] \# Y[0..1])$ */

Pour construire la solution (constitué par une boucle) qui soit le plus avantageux en nombre d'itération nous proposons l'invariant suivant :

Invariant :
/* $(0 \leq p \leq d \leq N-2) \wedge (\forall I : 0 \leq I < p \rightarrow X[I..I+1] \# Y[0..1]) \wedge (\forall I : d < I \leq N-2 \rightarrow X[I..I+1] \# Y[0..1])$ */

Q1. Initialiser p et d en respectant l'invariant **(1pt)**

Solution : P=0 ; d=N-2 ;

Q2. Proposer une condition pour la boucle **(1pt)**

Solution :

! ((Y[0] = X[p] && Y[1] = X[p+1]) && (Y[0] = X[d] && Y[1] = X[d+1]))

ce n'est pas grave si on utilise les opérateurs logique classique à la place de ! et de &&

Q3 Développer, en utilisant l'invariant, le corps de la boucle **(2pts)**

Solution :

{

/* $(0 \leq p \leq d \leq N-2) \wedge (\forall I : 0 \leq I < p \rightarrow X[I..I+1] \# Y[0..1]) \wedge (\forall I : d < I \leq N-2 \rightarrow X[I..I+1] \# Y[0..1])$ */
/* $\wedge \neg ((Y[0] = X[p] \wedge Y[1] = X[p+1]) \wedge (Y[0..1] = X[d..d+1]))$ */

```
if  (! (   (Y[0] = X[p] && Y[1] = X[p+1])  ) p++ ;
if  ( ! (Y[0] = X[d] && Y[1] = X[d+1])   )  d++ ;
```

/* $(0 \leq p \leq d \leq N-2) \wedge (\forall I : 0 \leq I < p \rightarrow X[I..I+1] \# Y[0..1]) \wedge (\forall I : d < I \leq N-2 \rightarrow X[I..I+1] \# Y[0..1])$ */

}

Q4 Proposer une variante **(1pt)**

Solution : N-p-d-1

Q5 Ecrire le programme commenté **(2 pts)**

Solution :

/* $(N>1) \wedge (\exists I : 0 \leq I \leq N-2 \wedge (Y[0] = X[I] \wedge Y[1] = X[I+1]))$ */

P=0 ; d=N-2 ;

/* $(0 \leq p \leq d \leq N-2) \wedge (\forall I : 0 \leq I < p \rightarrow X[I..I+1] \# Y[0..1]) \wedge (\forall I : d < I \leq N-2 \rightarrow X[I..I+1] \# Y[0..1])$ */

```
while (      ! (   (Y[0] = X[p] && Y[1] = X[p+1])  &&   (Y[0] = X[d] && Y[1] = X[d+1])   )  )
```

{

/* $(0 \leq p \leq d \leq N-2) \wedge (\forall I : 0 \leq I < p \rightarrow X[I..I+1] \# Y[0..1]) \wedge (\forall I : d < I \leq N-2 \rightarrow X[I..I+1] \# Y[0..1])$ */
/* $\wedge \neg ((Y[0] = X[p] \wedge Y[1] = X[p+1]) \wedge (Y[0..1] = X[d..d+1]))$ */

```
if (! ( (Y[0] = X[p] && Y[1] = X[p+1]) ) p++ ;
if ( ! (Y[0] = X[d] && Y[1] = X[d+1]) ) d++ ;
```

/* (0 ≤ p≤d≤N-2) ∧ (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]) */

}

/* (0 ≤ p≤d≤N-2) ∧ (Y[0..1] = X[p..p+1]) ∧ (Y[0..1] = X[d..d+1])
∧ (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]) */

B- Modifier le programme pour fournir la première position i **(1 Pt)**:

Solution :

i vérifie : (Y[0] = X[i] ∧ Y[1] = X[i+1])

(i est égale à la première position trouvée p ou d)

Q6- Reécrire la post condition du programme **(1 pt)**

/*

(0 ≤ p≤d≤N-2) ∧
(
((Y[0..1] = X[p..p+1]) ∧ (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]))

∨

((Y[0..1] = X[d..d+1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]))

)

***/**

Q7- Reécrire le programme commenté **(2 pts)**

```
(

/* (N>1) ∧ (∃I : 0≤I ≤N-2 ∧ (Y[0] = X[I] ∧ Y[1] = X[I+1]) ) */

      P=0 ; d=N-2 ;

/* (0 ≤ p≤d≤N-2)  ∧  (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]) */


while (      ! (   (Y[0] = X[p] && Y[1] = X[p+1]) ||   (Y[0] = X[d] && Y[1] = X[d+1])   )  )

            {

/* (0 ≤ p≤d≤N-2)  ∧  (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]) */
/*    ∧  ¬ ( (Y[0] = X[p] ∧ Y[1] = X[p+1])   ∨   (Y[0..1] = X[d..d+1]) )        */


                  p++ ;
                  d++ ;


/* (0 ≤ p≤d≤N-2)  ∧  (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]) */

            }

/* (0 ≤ p≤d≤N-2) ∧ (Y[0..1] = X[p..p+1])    ∨  (Y[0..1] = X[d..d+1])
∧  (∀ I : 0≤I< p → X[I..I+1] # Y[0..1]) ∧ (∀ I : d<I≤ N-2 → X[I..I+1] # Y[0..1]) */
```

II- Structure de données

Nous souhaitons implémenter le type EnsembleFini en Langage C.
Les ensemble que nous souhaitons traiter sont des ensembles d'entier de cardinal <= N (N est un constant dans le programme.

Nous rappelons que dans un ensemble une valeur ne peut apparaître plus d'une fois.

E = { 1, 4,0, 5}   **correct**
F = { 1, 4,0, 0, 5}   **incorrect**

Spécification fonctionnelle du type EnsembleFini :

Type EnsembleFini
Créer : → EnsembleFini /* on créé un ensemble vide
Ajouter : Entier X EnsembleFini → EnsembleFini /* on ajoute une nouvelle valeur
Union : EnsembleFini X EnsembleFini → EnsembleFini
Intersection : EnsembleFini X EnsembleFini → EnsembleFini

Pour implémenter le type EnsembleFini nous proposons une structure interne composée par 2 champs :

- Cardinal : entier naturel qui donne le nombre de valeurs qui appartennent à l'ensemble. 0 si l'ensemble est vide.
- Contenu : tableau de N cases. Seules les cases de 0..cardinal-1 sont significatives.
- Nous supposons que ajouter traite le cas de débordement de tableau

Q1. Ecrire en C la structure d'un ensemble sous forme de typedef etc.. **(2 pts)**

Solution :
typedef int t[N] contenu ;
typedef struct { int card ; contenu cont;} Ens_Fini ;

Q2. Ecrire en C l'opération union **(4 pts):**

Les opérations créer e t ajouter sont spécifiées on n'et pas obligé de les implémenter

On a besoin d'une fonction : int appart (int e , contenu t) qui retourne 1 si oui sino 0
E = { 1, 4,0, 5} F = { 0, 3, 5} G = Union(E,F) = { 1, 4,0, 5, 3}

Si on n'utilise pas cette fonction, on a intérêt à avoir son équivalent dans le code de union et d'intersection

Solution :

```
int appart (int e , contenu t )
    { int i , r ;
        r = 0 ;
        for( i = 0 ; i < t.card ; i++)  if (e == t.cont[i]{ r = 1; exit; }
```

```
        return r;
    }
```

```
Ens_Fini Union (Ens_Fini E , Ens_Fini F)
    { Ens_Fini G ; int i ;

    G = créer() ;
     for (i  = 0 ; i < E.card ; i++)
        if (! appart ( E.cont[i] ) ajouter (&G, E.cont[i] );
    for (i  = 0 ; i < F.card ; i++)  ajouter (&G, E.cont[i] );
     return G;

    }
```

Q3. Ecrire en C l'opération intersection (**3 pts):**

E = { 1, 4,0, 5}   F = { 0, 3, 5}   G = Intersecion(E,F) = { 0, 5 }

Solution :

```
Ens_Fini Intersection (Ens_Fini E , Ens_Fini F)
    { Ens_Fini G ; int i ;
    G = créer() ;
     for (i  = 0 ; i < E.card ; i++)
        if (appart ( E.cont[i] ) ajouter (&G, E.cont[i] );
    return G;
    }
```