--€€€

--POUR ESSAYER L'AGORA AVEC LE PROBLÈME DES CAMÉNÉONS
--(voir sujet d'examen de juin 2002)

-- Modifications apportées :
-- a) Remplacer Mail par Agora
-- b) La procédure Carnation.Mutation est appelée par chaque client (caménéon). On suppose que l'on
-- a la suite des appels Mutation(A, B); Mutation(B, A) avant que A ou B ne fasse de nouveaux
-- rendez-vous. Dans ce cas Mutation(B, A), le deuxième appel, est une opération nulle car les
-- caménéons viennent de prendre la même couleur. On suppose donc qu'on n'a pas une suite
-- d'appels comme : Mutation(A, B); Mutation(A, C); Mutation(B, A). Le programme est donc faux
-- par rapport au problème posé. Si on veut un programme exact, il faut remettre carnation dans
-- Agora.Rencontre et changer sa signature en Agora.Rencontre(X, Y : in out Integer).
-- Autre solution plus complexe : faire coopérer les deux caménéons A et B qui appellent chacun
-- Mutation.
-- c)Rendre la procédure Carnation.Mutation utilisable en concurrence par deux clients (caménéons)

-- €€

```
-- ************************                  paquetage Semaphores          **
package Semaphores is
      type Semaphore is limited private;
      procedure P(S : in out Semaphore); procedure V(S: in out Semaphore);
      procedure E0(S : in out Semaphore; Val: in natural);
private
      protected type Sem is
            entry P;
            entry V;
            procedure E0(Val: Natural);
      private
            Compteur : Natural := 0;
            Init : Boolean := False;
      end Sem;
      type Semaphore is new Sem;
end Semaphores;
------------------------------------------
package body Semaphores is
      -------------------------------
      protected body Sem is
            entry P when (Compteur >0 ) and (Init) is
            begin
                  Compteur := Compteur - 1;
            end P;
            entry V when Init is
            begin
                  Compteur := Compteur + 1;
            end V;
            procedure E0(Val : in Natural) is
            begin
                  Compteur := Val;
                  Init := True;
            end E0;
      end Sem;
      -----------------------------------
      procedure P(S : in out Semaphore) is
      begin S.P; end P;

      procedure V(S : in out Semaphore) is
      begin S.V; end V;
```

```
    procedure E0(S : in out Semaphore; Val: in natural) is
    begin S.E0(Val); end E0;
    ---------------------------------------
end Semaphores;
-- **********************                    paquetage Carnation     **
package Carnation is
    type Couleur is (Bleu, Jaune, Rouge);
    NbCameneons : Constant Integer := 50;
    type IdProc is mod NbCameneons; -- type modulo, range 0..NbCameneons-1
    function GetCouleur(X : in IdProc) return Couleur;  -- c'est la couleur de X
    procedure Mutation(X, Y : in out IdProc) ;
end Carnation;

with Semaphores; use Semaphores;
package body Carnation is
    -- table contenant la couleur de chaque caménéon
    LaCouleur : array(IdProc) of Couleur :=
                (IdProc'First => Couleur'First,
                IdProc'Last => Couleur'Last,
                others => Couleur'Succ(Couleur'First));
    -- le premier est Bleu, le dernier est Rouge, les autres sont Jaunes
    -- LaCouleur(IdProc'First) := Couleur'First; -- LaCouleur(1) = Bleu
    -- LaCouleur(IdProc'Last) := Couleur'Last; -- LaCouleur( Last) = Rouge
    -- le langage Ada permet de donner des attributs à un type numérique
    -- Couleur'First = Bleu; Couleur'Last = Rouge; Couleur'Succ(Bleu)= Jaune
    -- Couleur'Pos(Bleu) = 0; Couleur'Pos(Jaune) := 1; Couleur'Pos(Rouge) = 2
    -- Couleur'Val(0) = Bleu; Couleur'Val(1) = Jaune; Couleur'Val(2) = Rouge;

    Mutex : Semaphore; -- Mutation doit modifier les couleurs de façon cohérente
    procedure Mutation(X, Y : in out IdProc) is
    begin
        P(Mutex); -- car les caménéons X et Y peuvent l'appeler concurrement
        if LaCouleur(X) /= LaCouleur(Y) then
        LaCouleur(Y) := Couleur'Val(3 -
                                    Couleur'Pos(LaCouleur(X)) -
                                    Couleur'Pos(LaCouleur(Y)));
        LaCouleur(X) := LaCouleur(Y);
        end if;
        V(Mutex);
    end Mutation;

    function GetCouleur(X : IdProc) return Couleur is
    begin return LaCouleur(X); end GetCouleur;

begin E0(Mutex, 1); end Carnation;
--**************************                  paquetage Nom       **
with Carnation; use Carnation;

package Nom is
    procedure Unique(X : out IdProc);
end Nom;

with Semaphores; use Semaphores;

package body Nom is
    Mutex : Semaphore;
    Valeur : IdProc := IdProc'First;
    procedure Unique(X : out IdProc) is
    begin
    P(Mutex);
    X := Valeur; Valeur := IdProc'Succ(Valeur);
```

```
      V(Mutex);
      end Unique;
begin E0(Mutex, 1); end Nom;
--**************************** paquetage Agora
**
with Carnation; use Carnation;
with Semaphores; use Semaphores;

package Agora is
      procedure Rencontre(X: in Integer; Y : out Integer) ;
end Agorta;

package body Agora is
      -- définition du type de nom local qui sert, dans Agora, à bloquer et réveiller les clients.
      Max : constant := 18; -- nombre maximum de clients dans l'agora
      type IdLocal is mod Max;
      type Message is
            record
                  X : Integer; -- nom global du client
                  I : IdLocal;  -- nom local du client
            end record;
      -- ***                              tampon de messages dans le paquetage Requete      *****
      Package Requete is
            procedure Deposer(X : in Message);
            procedure Retirer(Y : out Message);
            procedure RetirerDeux(Y, Z : out Message);
      end Requete;
      Package body Requete is
            -- déclarations pour gerer le tampon
            -- << à faire pour chaque question A2 et A3 >>
            Nplein, Nvide, MutexProd, MutexCons : Semaphore;
            TailleMax : constant := 8; -- taille du tampon T
            type Index is mod TailleMax ;  -- valeurs des index du tampon T
            T : array(Index) of Message;
            Tete , Queue : Index := 0;
            procedure Deposer(X : in Message) is
            begin
            -- << à faire pour chaque question A2 et A3 >>
                  P(Nvide); P(MutexProd);
                  T(Queue) := X; Queue := Queue + 1; -- c'est fait modulo automatiquement
                  V(MutexProd); V(Nplein);
            end Deposer;
            procedure Retirer(Y : out Message) is
            begin
            -- << à faire pour la question A2>>
                  P(Nplein); P(MutexCons);
                  Y := T(Tete); Tete := Tete + 1 ;  -- c'est fait modulo automatiquement
                  V(MutexCons); V(Nvide);
            end Retirer;
            procedure RetirerDeux(Y, Z : out Message) is
            begin
            -- <<  à faire pour la question A3 >>
                  P(MutexCons); P(Nplein); P(Nplein); --demande le droit d'en prendre deux
                  -- en faisant porter la section critique sur les 2 demandes de messages, on évite ainsi
                  -- que, lorsqu'il n'y a que 2 messages, 2 serveurs s'approprient chacun 1 message
                  Y := T(Tete); Z := T(Tete + 1) ; Tete := Tete + 2 ;  -- c'est fait modulo automatiquement
                  V(MutexCons); V(Nvide);  V(Nvide); -- libère deux cases
            end RetirerDeux;
            -- autre solution possible
```

```
        -- declare
        --    Mutex : Semaphore;
        --    procedure RetirerDeux(Y, Z : out Message) is
        --    begin P(Mutex); Retirer(Y); Retirer(Z); V(Mutex); end RetirerDeux;
        --begin E0(Mutex, 1); end;
    begin                    --  *****   initialisations des sémaphores déclarés dans Requete**
    -- << à faire pour A2 et A3 ; mettre les valeurs initiales des sémaphores >>
        E0(Nvide, TailleMax) ; E0(Nplein, 0); E0(MutexProd, 1); E0(MutexCons, 1);
    end Requete;
    --  ********************  ******************
    package NomLocal is
        -- pour la prise et restitution d'un nom local unique
        procedure Prendre(I : out IdLocal);
        procedure Rendre(I : in IdLocal);
    end NomLocal;
    package body NomLocal is
        NumLibre : array(IdLocal) of Boolean := (others => True);
                                                -- au début tous les noms sont libres
        -- << 1 :  à faire pour la question A1 >>
        Mutex : Semaphore; --  contrôle de cohérence
        procedure Prendre(I : out IdLocal) is
        -- On n'appelle cette procédure que lorsqu'on sait qu'il y a un nom inutilisé
        begin
            I := IdLocal'First; --  on utilise le paramètre I comme un index de parcours.
            << 2 : à rendre cohérente  pour la question A1 >>
            P(Mutex);
            loop
                exit when NumLibre(I);   -- on a trouvé un nom non utilisé, on sort de la boucle
                I := I + 1;
            end loop;
            NumLibre(I) := False;  -- on note qu'on utilise le nom I
            << 3 : à rendre cohérente  pour la question A1 >>
            V(Mutex);
        end Prendre;
        procedure Rendre(I : in IdLocal) is
        begin
            << 4 : à rendre cohérente  pour la question A1 >>
            P(Mutex);
            NumLibre(I) := True;
            << 5 : à rendre cohérente  pour la question A1 >>
            V(Mutex); -- Mutex inutile si opération atomique
        end Rendre;
    begin                    --  *****   initialisations des sémaphores déclarés dans NomLocal**
    << 6 : à faire pour A1; mettre les valeurs initiales des sémaphores >>
        E0(Mutex, 1) ;
    end NomLocal;
    --  ********************  ******************
    Sem : array(IdLocal) of Semaphore; -- déclaration pour attente et envoi de signaux
    Complice : array(IdLocal) of Integer; -- Complice(I) contient le nom du complice de I
    --  contrôle de la cohorte pour que le nombre de clients dans l'agora ne dépasse jamais 18,
    Seuil : Semaphore; -- déclaré pour gérer la cohorte

    procedure Rencontre(X: in Integer; Y : out Integer) is
        I: IdLocal := 0; M : Message;
    begin
        -- contrôle à l'entrée pour qu'il n'y ait que 18 clients dans l'agora
        << 7 : à faire  pour la question A1 >>
        P(Seuil);
```

```
            NomLocal.Prendre(I);
            -- envoi de la requête M
            M.X := X; M.I := I;
            Requete.Deposer(M);  -- dépose sa demande
            -- X attend le signal envoyé au nom local I pour sortir de la procédure
            P(Sem(I));
            -- X lit le nom de son associé
            Y := Complice(I);
            -- X rend le nom local I
            NomLocal.Rendre(I);
            -- sortie contrôlée de l'agora pour assurer la présence de 18 clients au plus
            << 8: à faire  pour la question A1 >>
            V(Seuil);
      end Rencontre;
--  ******************  ******************
      task Serveur; task body Serveur is
            M, N : Message; A, B : Integer; J, K : IdLocal;
      begin
            loop
                  Requete.Retirer(M); Requete.Retirer(N);     -- il faut les requêtes de deux clients,
                  -- Requete.RetirerDeux(M,N);     -- question A3, remplace la ligne précédente
                  A := M.X; B := N.X;  -- récupérer les noms globaux
                  J := M.I; K := N.I;            -- récupérer les noms locaux
                  Complice(J) := B; Complice(K) := A;  -- enregistrer les noms des complices
                  V(Sem(J)); V(Sem(K));   -- envoyer les signaux à J et K
            end loop;
      end Serveur ;
begin                               --  *****    initialisations des sémaphores déclarés dans Agora**
      << 9 : pour la question A1 ; mettre la valeur initiale de Seuil >>
       E0(Seuil, Max); for I in IdLocal loop E0(Sem(I), 0); end loop;
end Agora;
    -- ********************        fin de l'annexe AGORA        ********************
-- ************************** procedure Main      **

with Nom; with Agora; with Carnation; use Carnation;
with Ada.Text_IO; use Ada.Text_IO;

procedure Main is

      task type Animal;
      Seconds : constant Float := 1.0;

      task body Animal is
            MaCouleur, Y : Couleur;
            Ego : IdProc;
            Lui : IdProc;
      begin
            Nom.Unique(Ego); -- acquiert un numero unique
            MaCouleur := GetCouleur(Ego);
            for I in 1..50 loop
                  --  ce delai simule une vie en dehors du mail
                   delay Duration((Float(Ego)/Float(I))*Seconds);
                  Agora.Rencontre(Integer(Ego), Integer(Lui)); --paramètres au bon type
                  Mutation(Ego, Lui);
                  Y:= GetCouleur(Ego); -- pour voir si elle a changé
                  if Y /= MaCouleur then
                        Put_Line("Le caménéon" & IdProc'Image(Ego) &" : "&
                              Couleur'Image(MaCouleur)& "=>" & Couleur'Image(Y ));
                        MaCouleur := Y;
```

```
                end if;
            end loop;
        end Animal;
        Cameneon : array(IdProc) of Animal;
begin null; end Main;
```