Université P. & M. Curie Examen
Licence Mention Informatique – L3 7 Janvier 2010
Programmation Concurrente (LI330) Durée : 2 heures

Les téléphones portables doivent être éteints et rangés dans vos sacs.
Le mémento Ada/C est le seul document autorisé pendant l'épreuve.

Le barème (sur 20) est donné à titre indicatif.
Il vous est conseillé de soigner votre copie et de rédiger des réponses claires (en particulier les programmes).
Toutes vos réponses doivent être dûment justifiées.
Lisez attentivement le sujet. Toute réponse hors sujet sera considérée comme fausse.
L'oubli des **`.all`** *(à bon escient) dans la gestion des pointeurs sera considéré comme une erreur.*

Exercice 1 – Anneau dynamique de processus (9 points)

Considérons un anneau dynamique de tâches (dynamique dans le sens où on peut ajouter de nouvelles tâches à l'anneau). La tâche principale comporte donc deux phases :

1. Construction d'un anneau de `Taille_Anneau_Defaut` nœuds,
2. Répétition de demandes d'affichages et d'insertions de nouveaux nœuds.

L'anneau doit toujours comporter un nœud d'identifiant 0. Le nœud 0 a un rôle particulier car c'est lui qui relaie toutes les demandes (création de nouveaux nœuds, demande d'affichage de l'anneau).

Nous disposons des déclarations suivantes :

```
-- Constantes pour le systeme
Taille_Anneau_Defaut : constant Positive := 5;

-- Declaration d'un noeud
type Noeud;
type A_Noeud is access Noeud;

task type Noeud (Id : Natural) is
   -- donner son successeur a un noeud
   entry Init (Succ : in A_Noeud);
   -- demander un parcours de l'anneau (les noeuds affichent leur identite)
   entry Parcourir;
   -- demander l'insertion d'un nouveau noeud
   entry Inserer (Qui : in A_Noeud);
end Noeud;
```

Nous nous intéressons dans un premier temps à la création de l'anneau. Cette création est réalisée par le programme principal d'après le code donné ci-après (les variables `Ptr`, `Ptr_Succ` et `Ptr_Last` sont de type `A_Noeud`).

```
-- Creation de l'anneau
for I in reverse  0 .. Taille_Anneau_Defaut - 1 loop
   Ptr := new Noeud (I * 2);
   if I /= Taille_Anneau_Defaut - 1 then
      Ptr.all.Init (Ptr_Succ);
   else
      Ptr_Last := Ptr;
   end if;
   Ptr_Succ := Ptr;
end loop;
Ptr_Last.all.Init (Ptr_Succ);
```

Question 1 – 1,5 points

Construisez le chronogramme associé aux actions permettant la création de l'anneau pour la valeur indiquée dans `Taille_Anneau_Defaut`. Vous indiquerez clairement l'identité des tâches créées au fur et à mesure.

Dans la suite, nous considérerons qu'une tâche de type `Noeud` déclare deux variables locales : `My_Succ`, de type `A_Noeud` (initialisée à **`null`**) et `Tmp_Node` également de type `A_Noeud`. Ces variables sont donc utilisables en cas de besoin dans les traitements associés à vos points d'entrées.

Question 2 – 1 point
Programmez le source du traitement de la requête `Init`.

Question 3 – 1,5 points
Programmez les actions que doit exécuter une tâche `Noeud` lorsqu'elle reçoit une requête `Parcourir`. Chaque nœud affiche son identité et demande à son successeur de s'afficher tant que l'anneau n'est pas complètement parcouru. C'est le programme principal qui demande, en premier, au nœud d'identité 0 de s'afficher.

On fait pour l'instant l'hypothèse que lors de l'insertion d'un nœud d'identité n, il ne peut pas y avoir déjà dans l'anneau un nœud de même identité. Lors de l'insertion du nœud n :

- si n est supérieur à toutes les identités des nœuds figurant déjà dans l'anneau, alors le nœud est inséré juste avant le nœud 0,
- sinon il est inséré entre le nœud de plus grande identité inférieure à n et le nœud de plus petite identité supérieure à n.

Une requête d'insertion est toujours émise par le programme principal à destination du nœud d'identité 0.

Question 4 – 2,5 points
Programmez les actions que doit exécuter une tâche `Noeud` lorsqu'elle reçoit une requête `Inserer`.

Pour un nœud donné, la requête `Init` doit être invoquée une fois et une seule, avant l'invocation de toute autre requête. La tâche peut ensuite être sollicitée pour traiter de manière répétitive des insertions de nœuds ou des affichages.

Question 5 – 1 point
Écrivez le corps d'une tâche de type `Noeud`, sans re-détailler les traitements liés aux différents points d'entrée.

Question 6 – 0,5 point
À quelle condition ce programme se termine-t-il ? Justifiez votre réponse.

Question 7 – 1 point
On considère maintenant qu'un nœud à insérer peut avoir une identité qui existe déjà. Dans ce cas, il ne faut pas procéder à l'insertion mais terminer la tâche que l'on devait rajouter. Comment procèderiez-vous pour traiter ce nouveau cas de figure (on demande les principes de la solution) ?

Exercice 2 – Le convoyeur (11 points)

Considérons un système de convoyeur chargé de transporter des objets identifiés par un identificateur unique (de type `positive`) depuis une *station de départ* vers une *station d'arrivée*. La figure 1 montre la structure de ce système.

Les objets sont déposés sur le convoyeur par une grue et extraits du convoyeur de l'autre coté par une autre grue.

Le convoyeur circule sur un rail passant au dessus d'un bras de mer. Comme le rail ne peut supporter qu'un poids limité, le convoyeur cesse d'accepter des objets dès que `Capacite_Convoyeur` (en Kg) est atteinte. Pour des raisons de volume, on ne peut non plus transporter plus de `Max_Objets_Convoyeur` par trajet.

Ainsi, pour piloter le système, il faut gérer le poids de chaque objet sur le convoyeur. Des ingénieurs ont analysé le système et proposent l'architecture suivante :

- Les objets sont équipés d'un contrôleur qui les pilote ; dans la suite du sujet, nous appellerons `Objet` le contrôleur associé à un objet ;
- Le convoyeur est également équipé d'un contrôleur dédié qui le pilote ; dans la suite du sujet, nous appellerons `Convoyeur` le contrôleur associé au convoyeur.

Les grues ne font que réagir aux sollicitations des objets et du convoyeur. Artificiellement, elles ne sont pas considérées comme des entités du système.

Pour développer les programmes de pilotage du système, on dispose des déclarations suivantes :

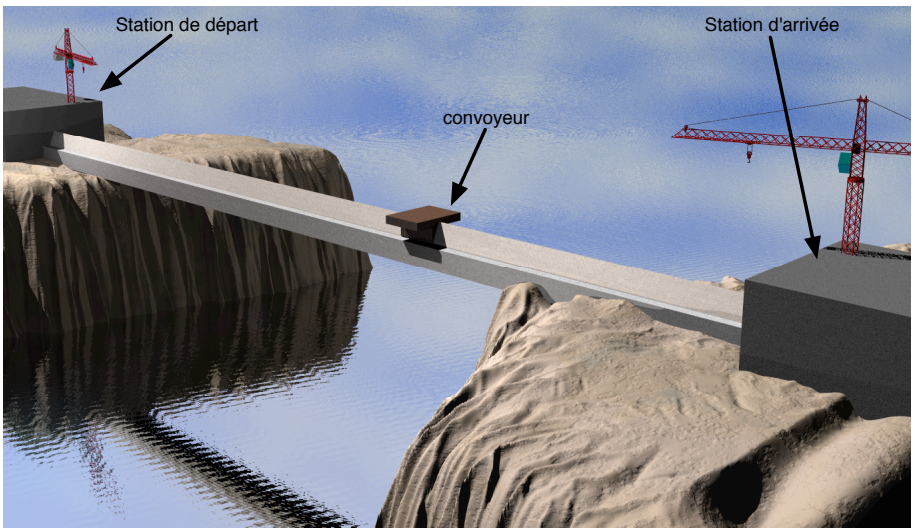

FIGURE 1 – Architecture du système

```
-- Constantes et autres
Capacite_Convoyeur    : constant Positive := 500;
Max_Objets_Convoyeur  : constant Positive := 6;
Taille_Tableau_Objets : constant Positive := 20;

-- Calculer un poids aleatoire
function Attribuer_Poids return Positive;

-- Declaration des Objets
type Objet;
type A_Objet is access Objet;

task type Objet (Id : Positive; Poids : Positive) is
   -- transmettre son pointeur a un objet
   entry Init (Me : in A_Objet);
   -- l'objet demande a etre extrait du convoyeur
   entry Extraire_Du_Convoyeur;
end Objet;

-- Declaration du convoyeur
task Convoyeur is
   -- Le convoyeur signale que l'objet peut-etre depose dans le convoyeur
   entry Deposer_Dans_Convoyeur (Poids : in Positive; Qui : in A_Objet);
end Convoyeur;
```

On considère que les objets sont déclarés dans un tableau comme suit

```
Tab_Objets : array (1 .. Taille_Tableau_Objets) of A_Objet;
```

Question 1 – 1,5 points

Écrivez le source du programme principal qui initialise `Tab_Objets` en créant un ensemble d'`Objet` d'identifiants `1` à `Taille_Tableau_Objets`, et qui transmet à chaque `Objet` un pointeur sur lui-même.

Le protocole suivi par chaque objet est le suivant :

- récupérer le pointeur sur lui-même ;
- envoyer une requête au convoyeur en transmettant son poids ;
- attendre que le convoyeur lui signale qu'il est arrivé à destination ;

Après l'exécution de ces opérations, l'objet se "termine".

Question 2 – 1,5 points

Écrivez le source du corps de la tâche `Objet`.

Question 3 – 1 point
Expliquez pourquoi la garde associée à l'**`accept`** d'un point d'entrée ne peut être fonction de l'un des paramètres de ce point d'entrée.

Nous allons maintenant nous intéresser au `Convoyeur`. Celui-ci effectue une suite de chargements. Un chargement se termine lorsque l'une des trois conditions suivantes est vérifiée :
- La capacité maximum du convoyeur en terme de poids avant chargement du nouvel objet est atteinte [1],
- Le nombre maximum d'objets sur le convoyeur est atteint,
- Aucun objet n'est arrivé dans les 10 dernières secondes.

Les variables locales suivantes sont déclarées dans le corps de la tâche `Convoyeur`.

```
Poids_Courant : Natural := 0;
Nb_Objet      : Natural := 0;
Objets_Inside : array (1 .. Max_Objets_Convoyeur) of A_Objet := (others => null);
```

Question 4 – 2 points
Écrivez la boucle permettant de charger le convoyeur (*i.e.* accepter les demandes des objets jusqu'à ce qu'un chargement soit complet).

Le convoyeur exécute de manière répétitive les actions suivantes : effectuer un chargement, se déplacer (faire un affichage), signaler aux objets transportés qu'ils sont arrivés à destination, puis revenir à son état initial.
Le convoyeur s'arrête dès qu'il détecte un chargement vide, avant d'effectuer son déplacement.

Question 5 – 2 points
Écrivez le source de la tâche `Convoyeur` (sans re-détailler les actions de chargement).

Nous souhaitons maintenant étendre le système à plusieurs convoyeurs, mais le rail qui traverse le bras de mer ne peut en supporter qu'un seul à la fois. Pour gérer l'accès au rail, on dispose d'un objet protégé `Acces_Pont` dont la spécification et le corps sont donnés ci-dessous :

```
protected Acces_Pont is
   procedure Sortir;
   procedure Entrer;
private
   Occupe : Boolean := false;
end Acces_Pont;

protected body Acces_Pont is
   procedure Sortir is
   begin
      Occupe := False;
   end Sortir;

   procedure Entrer is
   begin
      while Occupe loop
        delay 0.1;
      end loop;
      Occupe := True;
   end Entrer;
end Acces_Pont;
```

Question 6 – 2 points
Expliquez précisément ce qu'il se passe lorsque plusieurs convoyeurs demandent en même temps à accéder au pont.

Question 7 – 1 point
Modifiez l'objet protégé `Acces_Pont` pour garantir que l'accès au pont s'effectue correctement.

1. Il existe donc un seuil de tolérance qui permet le chargement d'un seul objet même si on dépasse le poids autorisé.