Université P. & M. Curie 2ème session
Licence Mention Informatique – L3 26 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.*

Préambule : il n'y a qu'un seul problème mais un certain nombre de questions sont indépendantes les unes des autres. Vous pouvez donc passer aux questions suivantes si vous "séchez" à un point du problème.

Exercice 1 – Vous ne verrez plus jamais les ascenseurs comme avant (20 points)

Considérons une batterie de `Max_Ascenseurs` ascenseurs dans un immeuble. Des usagers empruntent ces ascenseurs pour se rendre d'un étage de départ à un étage d'arrivée. Pour cela, ils appellent un ascenseur via une file de requêtes. Chaque requête sera traitée par l'un des ascenseurs libres.

Nous allons dans un premier temps traiter une version simplifiée de ce système dans lequel chaque ascenseur ne traite *qu'une seule requête à la fois*. Pour cela nous disposons des déclarations suivantes :

```
-- Constantes pour le système
Max_Requetes   : constant Positive := 5;
Max_Usagers    : constant Positive := 20;
Max_Ascenseurs : constant Positive := 3;
Max_Etages     : constant Positive := 30;

-- Types requis pour le système
type Usager;
type A_Usager is access Usager;

type Ascenseur;
type A_Ascenseur is access Ascenseur;

type Numero_Etages is range 0 .. Max_Etages;

type Requete is record
   Etage_Depart  : Numero_Etages;
   Etage_Arrivee : Numero_Etages;
   Qui           : A_Usager;
end record;

type Tab_Requetes is array (1 .. Max_Requetes) of Requete;

-- Déclaration des entités du système
task type Usager (Id : Positive; Depart, Arrivee : Numero_Etages) is
   -- permet a la tache de recuperer un pointeur sur elle meme
   entry Init (Self : in A_Usager);
   entry Entre (Ascenseur : in Positive);
   entry Sort;
end Usager;

task type Ascenseur (Id : Positive);

protected File_Requetes is
   entry Prendre_Requete (Depart, Arrivee : out Numero_Etages; Qui : out A_Usager);
   entry Deposer_Requete (Depart, Arrivee : in Numero_Etages; Qui : in A_Usager);
private
   Nb_Requetes : Natural := 0;
   Requetes    : Tab_Requetes;
end File_Requetes;
```

Nous nous intéressons à la programmation d'un `Usager` du système qui doit appeler l'ascenseur, entrer lorsqu'il y en a un qui arrive et sortir lorsqu'il se trouve à l'étage d'arrivée.

Question 1 – 2 points
Programmez le source du corps d'une tâche `Usager`.

On souhaite traiter les requêtes de manière ordonnée. Une nouvelle requête qui arrive doit donc être stockée dans la première case *libre* du tableau, et la requête à traiter est toujours prélevée dans la première case du tableau

Question 2 – 2 points
Programmez l'**`entry`** `Prendre_Requete` de `File_Requetes`.

Question 3 – 2 points
Programmez l'**`entry`** `Deposer_Requete` de `File_Requetes`.

Question 4 – 1 point
Donnez le code permettant de créer une batterie de `Max_Ascenseurs` Ascenseurs (Vous déclarerez la ou les variables que vous jugez nécessaire).

Nous donnons maintenant le code associé à un ascenseur simplifié :

```
task body Ascenseur is
   Etage_Courant    : Numero_Etages := 0;
   Etage_Dep_Usager : Numero_Etages;
   Etage_Arr_Usager : Numero_Etages;
   Quel_Usager      : A_Usager;
begin
   loop
       select
         File_Requetes.Prendre_Requete (Etage_Dep_Usager, Etage_Arr_Usager, Quel_Usager);
         Etage_Courant := Etage_Dep_Usager;
         Put_Line ((1 .. 2 * Id => ' ') & "Ouverture des portes de l'ascenseur" & Positive'Image (Id));
         Quel_Usager.all.Entre (Id);
         Put_Line ((1 .. 2 * Id => ' ') & "Fermeture des portes de l'ascenseur" & Positive'Image (Id));
         Etage_Courant := Etage_Arr_Usager;
         Put_Line ((1 .. 2 * Id => ' ') & "Ouverture des portes de l'ascenseur" & Positive'Image (Id));
         Quel_Usager.all.Sort;
         Put_Line ((1 .. 2 * Id => ' ') & "Fermeture des portes de l'ascenseur" & Positive'Image (Id));
      or
         delay 5.0;
         exit;
      end select;
   end loop;
   Put_Line ("plus de requete pour l'ascenseur " & Positive'Image (Id));
end Ascenseur;
```

Question 5 – 1 point
Peut-on remplacer l'alternative **`delay`** suivie du **`exit`** par une directive **`terminate`** ? Justifiez en 5 lignes maximum votre réponse.

Question 6 – 2 points
On considère un Usager initialisé allant de l'étage 3 à l'étage 6 et un ascenseur. Donnez le chronogramme correspondant au traitement de la requête.

Pour la suite du problème, on introduit les déclarations suivantes :

```
-- Les directions que peuvent prendre un ascenseur
type Direction is (Monter, Descendre);

-- Calcule la direction d'un trajet en fonction de l'étage de départ et de l'étage d'arrivée
function Calcule_Direction (Dep, Arr : in Numero_Etages) return Direction;
```

Mais la stratégie proposée dans la version simplifiée n'est pas acceptable car les ascenseurs ont une capacité supérieure à une personne. Il faudrait donc être capable de prendre plusieurs usagers. Pour cela, il faut modifier `File_Requetes` car on doit pouvoir extraire une requête concernant un étage donné et une direction donnée. Pour cela, on ajoute un nouveau service dont la spécification est donnée ci-après :

```
-- Recherche de la premiere requête dont l'étage de départ est Etage et dont la direction est la même que
-- Dir. Si cette requête existe, elle est décrite dans Qui, Depart et Arrivée. Dans le cas contraire,
-- Qui = null. Lorsqu'une requête est trouvée, elle est retirée de la file des requêtes.
procedure Prendre_Requete_Etage (Etage           : in Numero_Etages;
                                 Dir             : in Direction;
                                 Qui             : out A_Usager;
                                 Depart, Arrivee : out Numero_Etages);
```

Question 7 – 1 point

Expliquez pourquoi `Prendre_Requete_Etage` **doit** être une **`procedure`**.

Question 8 – 2 points

Écrivez le source de `Prendre_Requete_Etage` dans le corps du nouvel objet protégé `File_Requetes`.

On peut maintenant mettre en place la stratégie suivante qui optimise les déplacements de l'ascenseur en prenant plusieurs passagers quand ces derniers vont dans la même direction et sont sur le trajet d'une première requête. Cette stratégie est cyclique, et chaque cycle se déroule de la manière suivante :

- **Etape 1** : l'ascenseur récupère dans la file des requêtes une première demande d'usager. En fonction de la requête, il calcule la direction courante de l'ascenseur et se rend à l'étage de départ pour cet usager.
  Il permet ensuite à l'usager d'entrer et ajoute la requête correspondante dans la liste des requêtes locales (pour connaître l'étage de destination).

  Les étapes suivantes de la stratégie sont répétées à tous les étages que parcourt l'ascenseur dans la direction déterminée par la première requête et ce jusqu'à ce qu'il n'y ait plus d'usager dans l'ascenseur (NB : lorsqu'il n'est pas vide, l'ascenseur s'arrête à tous les étages).
- **Etape 2** : s'il y a des usagers dont la destination est l'étage courant, l'ascenseur leur signale qu'ils peuvent sortir.
- **Etape 3** : s'il y a, à l'étage courant, des usagers en attente dont la direction de déplacement correspond à celle de l'ascenseur, et tant que la capacité maximale de l'ascenseur n'est pas atteinte, on les fait entrer. Ceux qui ne peuvent pas entrer attendent le prochain passage d'un ascenseur (l'ordre de priorité des usagers correspond à l'ordre d'arrivée de leurs requêtes).
- **Etape 4** : si l'ascenseur n'est pas vide, il passe à l'étage suivant.

Si, après avoir traité les éventuelles requêtes d'usagers en attente à l'étage courant, l'ascenseur est vide, il entame un nouveau cycle.

Une tâche `Ascenseur` se termine lorsque l'ascenseur a attendu sans succès pendant 10 secondes qu'une nouvelle requête arrive dans la file du système.

Vous devez programmer cette nouvelle stratégie en considérant que vous disposez des déclarations suivantes qui sont locales à une tâche de type `Ascenseur` et que vous pourrez utiliser par la suite :

```
-- Nombre maximum d'usagers dans l'ascenseur
Capacite_Ascenseur : constant Positive := 4;

-- Variables locales
Etage_Courant      : Numero_Etages := 0;
Requetes_Locales   : array (1 .. Capacite_Ascenseur) of Requete;
Usagers_Dedans     : Natural       := 0;
D_U, A_U           : Numero_Etages;
Q_U                : A_Usager;
Direction_Courante : Direction;

-- Rajoute une requête dans la liste des requêtes gérées par l'ascenseur
procedure Ajouter_Requete_Locale (Dep, Arr : in Numero_Etages; Qui : in A_Usager);

-- Cherche dans la liste des requêtes gérées par l'ascenseur une requête qui correspond à l'étage Etg.
-- Si la requête est trouvée, Dep, Arr et Qui contiennent les informations la caractérisant.
-- Si aucune requête ne concerne Etg, Qui prend la valeur null
procedure Retirer_Requete_Locale (Etg : in Numero_Etages; Dep, Arr : out Numero_Etages; Qui : out A_Usager);

-- Ouvre les portes si elles ne sont pas déjà ouvertes
procedure Ouvrir_Portes;
```

```
-- Ferme les portes si elles ne sont pas déjà fermées
procedure Fermer_Portes;
```

Question 9 – 1,5 points

Programmez le corps d'une tâche `Ascenseur` en faisant figurer les étapes indiquées ci-dessus (mais sans les programmer). Vous indiquerez explicitement la condition de sortie d'un cycle de traitement et la condition de terminaison de la tâche.

Dans les réponses aux questions suivantes, vous devrez faire figurer les mouvements des portes des ascenseurs.

Question 10 – 2 points

Programmez l'étape 1 de la stratégie.

Question 11 – 1,5 points

Programmez l'étape 2 de la stratégie (débarquement des usagers).

Question 12 – 1,5 points

Programmez l'étape 3 de la stratégie (embarquement d'usagers dans la bonne direction).

Question 13 – 0,5 point

Programmez l'étape 4 de la stratégie (changement d'étage).