SAE 101 : La gestion d'un parc de voitures
- Troisième séance -

Objectifs :

- Implementation du paquetage *actions_parking*
- Travailler le menu de l'application

Nous allons continuer l'application en Ada pour la gestion d'un parc de voitures en utilisant les paquetages *voiture* et *parking*. Dans cette troisième séance, nous aimerons programmer :

- le paquetage *actions_parking* qui comporte un ensemble de nouveaux sous-programmes permettant la manipulation du *Type_Parking*
- un menu utilisateur pour manipuler toutes les actions du parking

Ainsi, vous allez construire les jeux de tests fonctionnels pour s'assurer que votre code satisfait le comportement attendu.

En ce qui concerne le dépôt de cette semaine : déposez les sources de votre projet Ada dans un dépôt Moodle à 18h30.

1 Programmer le paquetage *actions_parking*

1. Créez les deux fichiers du paquetage *actions_parking* (*actions_parking.ads* et *actions_parking.adb*).

   Nous allons maintenant à écrire encore quelques sous-programmes qui implémentent des nouveaux opérations de manipulation du *Type_Parking*. N'oubliez de proposer (en parallèle avec l'implémentation des opérations) un programme principal *test_actions_parking.adb* qui assure le comportement attendu des sous-programmes.

2. Écrire le sous-programme *nb_places_disponibles* qui calcules les places disponibles du parking *p* de type *Type_Parking*.

3. Écrire le sous-programme *obtenir_tarif* qui calcule la tarif que le parking devrait appliquer à une durée de stationnement *d*. Voici les différentes tarifs :

Durée (secondes) :	Tarif
0 <d≤3600 :	0.16
3600<d≤7200 :	0.20
7200<d≤10800 :	0.32
d>10800 :	0.6

4. Écrire le sous-programme *obtenir_voiture_plus_longue_duree* qui retourne la voiture qui a la plus longue durée de stationnement.

5. Écrire le sous-programme *obtenir_plus_grand_num_place* qui retourne la voiture qui a la plus grande numéro de place.

1.1 Menu

Écrire un nouveau programme principal *menu.adb* qui propose un menu à l'utilisateur dans lequel il pourra tester toutes les opérations du parking (paquetage *parking* et *actions_parking*). Dans ce menu, l'utilisateur peut choisir une fonctionnalité ou quitter l'application. Voici la liste d'operations :

0 :	Remplissage du parking manuellement à partir du clavier
1 :	Affichage de la liste de voitures du parking
2 :	Trouver une voiture dans le parking
3 :	Ajouter une voiture dans le parking
4 :	Ajouter une voiture dans le parking dans une position donnée
5 :	Supprimer une voiture dans le parking
6 :	Supprimer une voiture dans le parking dans une position donnée
7 :	Calculer le nombre de places disponibles
8 :	Obtenir la voiture avec la plus longue durée
9 :	Obtenir la tarif pour une durée
10 :	Obtenir le numero de la place la plus grande

En annexe 1, vous trouverez une proposition de solution pour implémenter ce menu. Nous vous demandons également de faire en sorte que ce menu soit itératif en utilisant *la boucle jusqu'à* (cf. annexe 2).

1.2 Lecture du parking à partir d'un fichier

Écrire le sous-programme *lire_parking_from_file* qui lit les informations du parking *p* de type *Type_Parking* à partir d'un fichier *filename*. Utiliser le fichier disponible sur Moodle *parking.txt* pour regarder comment les informations sont stockées dans le fichier.

Utilisez le cours disponible sur Moodle pour la manipulation de fichiers !

Tester le sous-programme *lire_parking_from_file* dans le programme principal *test_actions_parking* et ajouter l'operation dans le menu utilisateur.

2 Annexe 1 : Le menu

Si le choix d'un utilisateur correspond à une valeur d'une valeur entière, vous pouvez utiliser l'instruction case. Cette instruction est particulièrement appropriée pour des choix multiples. Comme son nom l 'indique, elle permet de traiter différents cas. Vous trouverez ci-dessous un exemple :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Check_Direction is
   N : Integer;
begin
   loop
      Put ("Enter an integer value: ");  -- Puts a String
      Get (N);  --  Reads an Integer
      Put (N);  --  Puts an Integer
      case N is
         when 0 | 360 =>
            Put_Line (" is due north");
         when 1 .. 89 =>
            Put_Line (" is in the northeast quadrant");
         when 90 =>
            Put_Line (" is due east");
         when 91 .. 179 =>
            Put_Line (" is in the southeast quadrant");
         when 180 =>
            Put_Line (" is due south");
         when 181 .. 269 =>
            Put_Line (" is in the southwest quadrant");
         when 270 =>
            Put_Line (" is due west");
         when 271 .. 359 =>
            Put_Line (" is in the northwest quadrant");
         when others =>
            Put_Line (" Au revoir");
            exit;
      end case;
   end loop;
end Check_Direction;
```

3 Annexe 2 : La boucle *la boucle jusqu'à ce que*

Il existe une boucle nommé "jusqu'à ce que" qui permet de ne faire le test qu'après avoir exécuté le corps de la boucle. Cette forme permet de construire une structure répétitive dans laquelle la condition de rebouclage est vérifiée à la fin : on est donc certain d'exécuter au moins une fois le bloc d'instruction à répéter. Voir un exemple ci-dessous :

```
function factorielle (n : in Integer) return Integer is
   counter : Integer:=n;
   result : Integer :=1;
begin

   loop
     result:= result * counter;
     counter   := counter - 1;
     exit when counter = 0;
   end loop;

   return result;
end factorielle;
```