Un gestionnaire de location de vélos semblable à la 'Vélo toulouse'

Tchomgue Mieguem Ivan Brunel
Groupe D

23 janvier 2012

Table des matières

Première partie

Présentation générale

1 Introduction

Le système de location de vélos permet à des utilisateurs abonnés de pouvoir utiliser un vélo disponible dans une station pour se déplacer dans la ville.
L'objectif de ce projet est la réalisation d'un logiciel permettant d'assurer la création et la suppression de stations, la gestion des utilisateurs abonnés et la location de vélos.

2 Spécifications générales

Les structures de données dynamiques (c'est-à-dire de taille non fixée à l'avance) devront être implantées par des pointeurs et non par des tableaux ou des matrices.

Un vélo identifié par un numéro unique dans la totalité du parc à vélos, est pris dans une station et devra être restitué dans une autre (ou dans la même) station après utilisation. Dans tous les cas, un vélo ne pourra être pris que s'il est disponible dans la station et ne pourra être restitué que si la station dispose d'une place de parking libre. Un vélo ne peut être affecté qu'à un utilisateur au plus à un instant donné.

Les stations de vélos sont identifiées par un numéro. Elles comprennent un nombre fixé de vélos, qui peut être différent d'une station à l'autre. Pour une station donnée,on mémorisera l'ensemble des stations distantes de moins d'un kilomètre. De plus, dans une station, il est possible de connaître l'ensemble des vélos disponibles ainsi que les stations les plus proches (moins d'un kilomètre) disposant de vélos ou de places de parking pour ranger les vélos.

Les utilisateurs abonnés sont décrits par un numéro, un code postal. Ils ont un crédit de 3 Euros à l'origine ainsi qu'une caution de 150 Euros utilisée en cas de détérioration, de perte ou de vol de vélo. Le crédit est débité de 50 centimes par demi heure d'utilisation au delà de la première demi-heure qui est gratuite. Lorsque ce crédit est nul, un utilisateur ne peut prendre de vélo si son compte n'est par crédité.

Deuxième partie

Conception et codage

3 Identification des fonctions de ce système

Le système de location de vélos doit être géré par un administrateur. Afin de séparer l'interface administrateur de celui des utilisateurs, les tâches reservées à l'administration seront protégées par un mot de passe.

Pour mettre en place ce systéme, l'administrateur a essentiellement besoin de pouvoir :

1. Créer des stations, en respectant la spécification générale mentionnée plus haut.
2. Détruire une ou plusieurs stations s'il le juge nécessaire.
3. Consulter la base de données afin de suivre l'évolution de chaque abonné.

Les utilisateurs quant à eux, doivent au préalable s'abonner pour bénéficier des services qu'offre chaque station mise en place.
Une fois abonné, chaque utilisateur pourra :

1. Prendre un vélo dans une station, si son crédit est suffisant.
2. Restituer ce vélo dans une station disposant de place de parking libre.
3. Suivre l'évolution de son compte :
   - consulter son credit et sa caution
   - avoir la possibilité de créditer son compte
   - savoir quel est le numéro de vélo qu'il possède
4. Avoir des informations sur les autres stations à savoir :
   - localisation des stations
   - liste des stations disposant de vélos ou de places de parking pour ranger les vélos
   - liste des stations distantes de moins de X mètres d'une station donnée.

Pour chaque point précédent on peut faire correspondre les fonctions suivantes :

- `creeStation`
- `enleveStation`
- `infoBaseDonnee`
- `abonnement`
- `prendreVelo`
- `restituerVelo`
- `monCompte`
- `miniCarte`
- `parkingsLibre`
- `stationDistanteXm`

3.1 Définition des types

A partir des informations de la section 2 de la page 2, on peut définir les types suivants :

On définit le type `listeVelo` comme suit :

```
type noeud;
type listeVelo is access noeud;
type noeud is record
   val:integer;                --numéro du vélo
   suivant:listeVelo;
end record;
```

velo.py

On définit les types `coordoonnees, station et reseauStation` comme suit :

```
type coordonnees is record
    absc:float;                --abscisse
    ordo:float;                --ordonnée
end record;

type station is record
  num:integer;                 --numéro de la station
  nbVelos:integer;             --nombre de places du parking à vélos
  coordo:coordonnees;          --coordonnées de la station
  placeLibre:boolean;          --peut-on ranger un vélo?(oui/non)
  VelosDispo:listeVelo;        --ensemble des vélos dans le parking
  stations_1_km:reseauStation;--liste des stations à la ronde
end record;

type noeud2;
type reseauStation is access noeud2;
type noeud2 is record
  val:station;
  suivant:reseauStation;
end record;
```

station.py

Enfin on définit les types `utilisateur et listeUtilisateur` comme suit :

```
type utilisateur is record
  num:integer;                 --numéro
  codePostal:integer;          --code postale
  credit:float;                --son crédit
  caution:integer;             --sa caution
  veloPris:listeVelo;          --vélos en sa possession
  tempsprise:integer;          --temps(en s) lors de la prise d'un vélo
end record;

type noeud;
type listeUtilisateur is access noeud;
type noeud is record
  val:utilisateur;
  suivant:listeUtilisateur;
end record;
```

user.py

3.2 Définition des paquetages

A partir des types que l'on vient de définir, on peut déjà identifier trois grands groupes de paquetages :

- le paquetage `p_velo` qui contiendra les fonctions liées aux vélos
- le paquetage `p_station` qui contiendra les fonctions liées aux stations
- le paquetage `p_utilisateur` qui contiendra les fonctions liées aux utilisateurs

On désigne par `T` un type qui peut être `listeVelo`, `reseauStation` ou `listeUtilisateur`

Dans chacun de ces paquetages, il est nécessaire d'avoir les fonctions suivantes :

```
------------------------------------------------------------------------------
--semantique:fonction listevide cree une liste vide
--parametres: aucun
--pre-condition: aucune
--post-condition: est_vide (l) vaut vrai
--type retour: T
--exception: aucune
function listevide return T;
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--semantique: fonction est_vide teste si une liste l de type T est vide
--parametres: l donnee type T
--type retour: booleen
--pre-condition: aucune
--post-condition: aucune
--exception: aucune
--tests:
  --l est le pointeur nul
  --l est une liste ayant au moins un élément
function est_vide (l:in T) return boolean;
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--semantique: enlever un element e de la liste l (liste vide ou non vide)
--parametres: l in out T
--e in integer
--pre-condition: aucune
--post-condition: e nappartient pas à la liste
--exception: aucune
--tests:
  --enlever un élément qui nappartient pas à la liste
  --enlever un élément à une liste vide
  --enlever un élément en tete de liste
  --enlever un élément ayant une position quelconque dans la liste
procedure enlever(l:in out T; e:in integer);
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--semantique: procedure inserer_en_tete
--insere en tete de la liste l de type T lelement nouveau
--parametres: l in out T
--nouveau in Q
--si T est de type reseauStation,Q est de type station
--si T est de type listeUtilisateur,Q est de type utilisateur
--si T est de type listeVelo, Q est de type integer (le numéro du vélo)
--pre-condition: aucune
--post-condition: nouveau appartient à la liste l
--exception: aucune
--tests:
  --inserer en tete d'une liste vide
  --inserer en tete d'une liste quelconque
procedure inserer_en_tete(l:in out T; nouveau:in Q);
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--semantique: fonction rechercher recherche si e appartient à la liste l
--de type T et retourne son adresse ou alors
--retourne null si la liste est vide ou si e n'appartient pas à la liste
```

```
--parametres: l in T, la liste où l'on recherche l'element.
--e in integer,l'element qu'on recherche
--type-retour: T
--pre-condition: aucune
--post-condition: aucune
--exception: aucune
--tests:
  --rechercher si un élément appartient à la liste vide
  --rechercher si un élément appartient à une liste ne contenant pas
  --cet élément
  --rechercher si un élément appartient à une liste quelconque
function rechercher(l:in T; e:in integer) return T;
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--semantique:procedure afficher affiche les elements de la liste l
--parametres: l in T, la liste dont on affichera les elements
--pre-condition: aucune
--post-condition: aucune
--exception: aucune
--tests:
  --afficher les éléments d'une liste vide
  --afficher les éléments d'une liste quelconque
procedure afficher(l:in T);
------------------------------------------------------------------------------


------------------------------------------------------------------------------
--semantique:fonction longueur,calcule la taille d'une liste l de type T
--parametre:
--l:in T
--pre-condition aucune
--post-condition aucune
--type de retour integer
--exception:aucune
--tests:
  --longueur d'une liste vide
  --longueur d'une liste quelconque
function longueur(l:in T) return integer;
------------------------------------------------------------------------------
```

generic.py

3.2.1 Le paquetage `p_velo`

En plus des fonctions du listing generic.py de la sous-section 3.2, page 4 où le type `T` represente ici le type `listeVelo`, nous aurons aussi besoin de générer des vélos pour les ranger dans les parkings des stations qu'on créera. Pour ce faire, nous utiliserons une fonction qui nous permettra de :

1. Créer le nombre de vélos qu'on souhaite
2. Les ranger dans le parking d'une station donnée
3. Ajouter les vélos générés à la base de donnée des vélos.

Nommons la `genereVelos`. Voici la spécification de cette fonction :

```

--semantique:procedure genereVelos
--cree une liste de velos pour un parking
```

```
    --parametres:
    --nbrevelo,in integer,nombre de vélos à  générer
    --databaseVelo, in out listeVelo, base de données des vélos
    --velosGenere,out listeVelo,la liste des vélos générés
    --preconditions:
    --aucune
    --postcondition:
    --la base de donnée des velos a nbreVelo en plus.
    --nbreVelo ont été généré
    --exception:aucune
    --tests:
      -- generer zero vélo
      -- generer n vélos
    procedure genereVelos (nbreVelo:in integer; databaseVelo:
    in out listeVelo; velosGenere:out listeVelo);
```

p_velo.py

3.2.2 Le paquetage `p_station`

En plus des fonctions du listing generic.py de la sous-section 3.2, page 4 où le type `T` represente ici le type `reseauStation`, nous aurons besoin des fonctions :

1. `rechercher` qui recherche si une station existe a un emplacement donné, afin d'éviter de construire deux stations au même endroit.
2. `stationDistanteX` qui renvoie les stations distantes de moins de X mètres d'une station donnée. Ceci dans le but de connaître la distribution des stations dans le réseau.
3. `afficheStationLibre` qui affiche les stations disposant de places de parking pour ranger les vélos.
4. `statStation` qui permet d'avoir un compte rendu détaillé de toutes les stations du réseau.

Voici les spécifications de ces fonctions :

```

    --sémantique: fonction rechercher recherche si la station
    --de coordonnée(x,y) appartient à la liste
    --retourne son adresse
    --ou null si la liste est vide ou si e nappartient pas à  la liste
    --paramètres:
      --l donnée type reseauStation
      --x,y in float,coordonnée recherchées
    --type-retour: reseauStation
    --pré-condition: aucune
    --post-condition: l'adresse de l'élément recherché est retourné
    --exception: aucune
    --tests:
      --recherche d'une station déja créee
      --recherche d'une station inexistante
    function rechercher(l:in reseauStation; x:in float;y:in float) return
    reseauStation;
    ------------------------------------------------------------------------

    ------------------------------------------------------------------------
    --sémantique: fonction stationDistanteX
    --retourne l'ensemble des stations distantes de moins de Xm
    --d'une station result donnée hormis la station result elle même
```

```
    --parametres:
      --databaseStation,in reseauStation, la base de données des stations
      --X,in float,la distance souhaitée
      --result:in station,la station de reference
    --pre-condition:aucune
    --post-condition: les stations distantes de moins de Xm sont retournées
    --type de retour: reseauStation
    --exception:aucune
    --tests:
      --la distante X vaut 0, aucune station n'est retournée
      --la base de donnée r est vide, on retourne null
    function stationDistanteX (databaseStation:in reseauStation;
    X:in float;result:in station) return reseauStation;
    ------------------------------------------------------------------------


    ------------------------------------------------------------------------
    --semantique:procedure afficheStationLibre
    --affiche les stations du reseau r ayant une place de parking libre
    --parametre:
      --r,in reseauStation
    --précondition:aucune
    --postcondition:les numéros de stations sont affichés
    --exception:aucune
    --tests:
      --r est vide
      --r ne contient aucune station libre
      --r a au moins une station libre
    procedure afficheStationLibre(r:in reseauStation);
    ------------------------------------------------------------------------


    ------------------------------------------------------------------------
    --sémantique:procedure statStation
    --donne un compte rendu concernant chacune des stations du reseau
    --coordonnées, vélos,le numéro etc
    --parametre:
      --r:in reseauStation,la base de donnée des stations
    --précondition:aucune
    --postconditon:le compte rendu de chaque station est affiché
    --exeption:aucune
    --tests:
      --la base de donnée est vide
      --la base de donnée a au moins une station
    procedure statStation (databaseStation:in reseauStation);
    ------------------------------------------------------------------------
```

p_station.py

Nous avons défini les types `coordoonée, station, reseauStation` plus haut. Tout au long de notre programme, nous aurons sûrement besoin de mettre à jour les champs de ces enregistrements, et/ou de recupérer leurs valeurs (pour l'affichage par exemple).

Nous définissons alors les fonctions de :

Mise à jour notées `set«NomduChamp»`. Elles permettront de modifier la valeur du champ `«NomduChamp»` d'un enregistrement donné.

Récupération notées `recup«NomduChamp»`. Elles permettront simplement de récupérer la valeur du champ `«NomduChamp»` d'un enregistrement. Ceci dans le but de réaliser des instructions d'affichage, ou de calcul.

Voici la spécification de ces fonctions :

```
1
   --sémantique:fonction recupNomduChamp
3  --retourne le champ NomduChamp d'un enregistrement donné
   --parametres:
5  --s, in (type de l'enregistrement notons le Q), enregistrement dont on
   --souhaite recupérer le champ
7  --precondition:aucune
   --postcondition:on dispose du champ souhaité
9  --type de retour: le type de l'enregistrement récupéré (notons le R)
   --exception:aucune
11 --tests:
   function recupNomduChamp (s:in Q) return R;
13 ------------------------------------------------------------------------------


15 ------------------------------------------------------------------------------
   --procedure setNomduChamp
17 --met à jour le champ NomduChamp en modifiant sa valeur
   --parametres:
19 --s:in out Q(type de l'enregistrement), l'enregistrement en question
   --val:in R(type du champ), la valeur avec laquelle
21 --on souhaite mettre à jour
   --précondition aucune
23 --postcondition le champ NomduChamp de l'enregistrement s a pour
   --nouvelle valeur, val
25 --exception:aucune
   procedure setNomduChamp (s:in out Q; valeur:in R);
27 ------------------------------------------------------------------------------
```

getters_and_setters.py

Pour mieux étayer, le fonctionnement de ces fonctions prennons les exemples suivant :

1. Modifier les coordonnées d'une station (ici on modifie le champ `coordo` de l'enregistrement `station`)
2. Récupérer la liste de vélos disponibles d'une station (ici on récupère le champ `velosDispo` de l'enregistrement `station`)
3. Modifier le numéro d'une station. (Une autre valeur est attribuée au champ num de l'enregistrement `station`)

On obtient alors :

```
                    1.MODIFICATION DES COORDOONNEES
--procedure setCoordo
--met à jour le champ coordo en modifiant sa valeur
--parametres:
--s:in out station, la station sur laquelle on agit
--x:in float, abscisse
--y:in float, ordonnée
--précondition aucune
--postcondition le champ coordo a pour valeur (x,y)
--exception:aucune
-------------------------------------------------------------------------
procedure setCoordo(s:in out station; x:in float; y:in float) is
begin
     s.coordo:=(x,y);
end setCoordo;

                    2.RECUPERATION DES VELOS DISPONIBLES
--sémantique:fonction recupVelosDispo
--retourne la liste des vélos diponibles d'une station donnée
--parametres:
--s, in station
--precondition:aucune
--postcondition:on dispose du champ souhaité
--type de retour: listeVelo)
--exception:aucune
-------------------------------------------------------------------------
function recupVelosDispo (s:in station) return listeVelo is
begin
   return s.VelosDispo;
end recupVelosDispo;

                    3.MODIFICATION DU NUMERO
-------------------------------------------------------------------------
--procedure setNum
--met à jour le champ num en modifiant sa valeur
--parametres:
--adrS:in reseauStation, l'adresse de la station sur laquelle on agit
--valeur:in integer, la valeur avec laquelle on modifie le champ num
--précondition aucune
--postcondition le champ num a pour valeur, valeur
--exception:le champ est nul
-------------------------------------------------------------------------
procedure setNum(adrS:in reseauStation; valeur:in integer) is
begin
     if adrS=null
     then
        raise exception_adrS;
     else
       adrS.all.val.num:=valeur;
     end if;
end setNum;
```

exemples.py

3.2.3 Le paquetage `p_utilisateur`

En plus des fonctions du listing generic.py de la sous-section 3.2, page 4 où le type `T` represente ici le type `listeUtilisateur`, nous aurons besoin des fonctions :

1. `rechercher` qui permettra d'effectuer une recherche selon un critère précis. Selon le critère précisé elle cherchera soit la boîte postale d'un utilisateur soit son numéro d'identifiant.
2. `statUser` qui permet d'avoir un compte rendu détaillé de tous les utilisateurs abonnés.

Voici les spécifications de ces fonctions :

```
1
    --sémantique: fonction rechercher
3   --recherche si e appartient à la liste l
    --retourne son adresse
5   --ou null si la liste est vide ou si e n'appartient pas à la liste
    --paramètres: l donnée type listeUtilisateur
7   --e donnée type entier
    --precision:in string, precision du champ de recherche(num,codePostal)
9   --type-retour: listeUtilisateur
    --pré-condition: aucune
11  --post-condition: aucune
    --exception: aucune
13  --tests:
        --rechercher un numéro,ou un code postal inexistant
15      --rechercher un numéro figurant dans la base de donnée
        --rechercher avec une precision invalide
17      --(dans ce cas,la fonction n'effectuera pas de recherche)
    function rechercher(l:in listeUtilisateur; e:in integer;
19  precision:in string) return listeUtilisateur;
    ------------------------------------------------------------------------
21
    ------------------------------------------------------------------------
23  --sémantique: fonction statUser
    --donne un compte rendu de chacun des abonnés
25  --paramètres:
    --databaseUser,in listeUtilisateur, base de données utilisateurs
27  --pré-condition: aucune
    --post-condition: aucune
29  --exception: aucune
    --tests:
31      --la base de donnée est vide
        --la base de donnée a au moins un utilisateur
33  procedure statUser(databaseUser:in listeUtilisateur);
    ------------------------------------------------------------------------
```

p_utilisateur.py

Les fonctions du listing getters_and_setters.py de la sous section 3.2.2 de la page 7 seront aussi utilisées ici. Il suffit de faire correspondre à Q et R les types adéquats figurant dans le paquetage p_utilisateur

3.3 Spécifications des fonctions principales

Nous possédons à présent d'outils nécessaires pour spécifier les fonctions principales définies à la section 3 de la page 3. Pour rappel, ces fonctions sont :

- `creeStation`
- `enleveStation`
- `infoBaseDonnee`
- `abonnement`
- `prendreVelo`
- `restituerVelo`
- `monCompte`
- `miniCarte`
- `parkingsLibre`
- `stationDistanteXm`

Voici les spécifications de ces fonctions.

```
--sémantique: procedure creeStation
--cree une nouvelle station
--parametres:
--databaseStation,in out reseauStation,base de donnée des stations
--databaseVelo,in out listeVelo,base de données des vélos déja créés
--les numeros des velos sont générés automatiquement
--chaque vélo a un numéro unique
--on a besoin de cette base de données pour eviter les répétitions
--préconditions:
--le numero de la station est unique
--postcondition:
--le reseau a une station de plus
--exception:aucune
procedure creeStation(databaseStation:in out reseauStation;
databaseVelo:in out listeVelo);
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--sémantique: procedure enleveStation
--enleve une station
--parametres:
--databaseStation,in out reseauStation,base de donnée des stations
--préconditions:
--la base de données des stations n'est pas vide
--postcondition:
--le reseau a une station en moins
--exception:aucune
 procedure enleveStation(databaseStation:in out reseauStation);
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--sémantique: procedure infoBaseDonnee
--affiche le statut des stations et des abonnés du réseau
--parametres:
--databaseStation,in reseauStation,base de donnée des stations
--databaseUser,in listeUtilisateur,base de donnée des abonnés
--préconditions:
--aucune
--postcondition:
--les informations sur les stations et sur les abonnés sont affichés
--exception:aucune
procedure infoBaseDonnee (databaseStation:in reseauStation;
databaseUser:in listeUtilisateur);
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--semantique:procedure abonnement
--ajoute un utilisateur à la base de données utilisateurs.
--parametres:
--databaseUser,in out listeUtilisateur, base de données
--preconditions:
--avoir un numéro valide (on suppose qu'un numéro est valide s'il
--n'a pas encore été utilisé,donc ne figure pas dans la base dedonnées)
--avoir une caution de 150 euros et un credit de 3 euros
--postcondition: un utilisateur est ajouté à  la base de données.
--exception: aucune
procedure abonnement (databaseUser:in out listeUtilisateur);
-------------------------------------------------------------------------------
```

```
60
   ------------------------------------------------------------------------
62 --semantique:procedure prendreVelo
   --autorise un abonné à  prendre un vélo
64 --parametres:
   --databaseVelo,in out listeVelo, base de données des vélos
66 --databaseUser,in listeUtilisateur, base de données des utilisateurs
   --databaseStation:in reseauStation,base de données des stations
68 --preconditions:
   --avoir des identifiants valides (on suppose qu'ils sont valides s'ils
70 --figurent dans la base de données)
   --avoir un credit suffisant et une caution de 150 euros
72 --velo disponible dans la station
   --postcondition: le vélo est enlevé de la station
74 --le numero de velo s'ajoute à la liste de vélos pris par l'utilisateur
   --exception: aucune
76 procedure prendreVelo(databaseVelo:in out listeVelo;
      databaseUser:in listeUtilisateur;databaseStation:in reseauStation);
78 ------------------------------------------------------------------------

80 ------------------------------------------------------------------------
   --sémantique procedure restituerVelo
82 --restitue un vélo à  une station
   --parametres:
84 --databaseVelo,in out listeVelo, base de données des vélos
   --databaseUser,in listeUtilisateur, base de données des utilisateurs
86 --databaseStation:in reseauStation,base de données des stations
   --preconditions:
88 --la station dispose d'au moins une place libre
   --postcondition:
90 --la station a un vélo en plus
   --le credit est débité en fonction du temps passé avec le vélo
92 --exception:aucune.
   procedure restituerVelo(databaseVelo:in out listeVelo;
94    databaseUser:in listeUtilisateur; databaseStation:in reseauStation);
   ------------------------------------------------------------------------
96

   ------------------------------------------------------------------------
98 --semantique:procedure monCompte
   --autorise un abonné à  consulter les détails de son compte
100 --parametres:
   --databaseUser,in listeUtilisateur, base de données des utilisateurs
102 --preconditions:
   --avoir un numéro valide (on suppose qu'un numéro est valide s'il
104 --figure pas dans la base de données)
   --postcondition: les infos sont affichés
106 --exception: aucune
   procedure monCompte(databaseUser:in listeUtilisateur);
108 ------------------------------------------------------------------------

110 ------------------------------------------------------------------------
   --semantique:procedure miniCarte
112 --affiche les stations du réseau ainsi que leurs coordoonées
   --parametres:
114 --databaseStation,in reseaustation, base de données des stations
   --preconditions: aucune
116 --postcondition: les infos sont affichés
   --exception: aucune
118 procedure miniCarte(databaseStation:in reseauStation);
   ------------------------------------------------------------------------
```

```
120
    ------------------------------------------------------------------------
122 --semantique:procedure parkingsLibre
    --affiche les stations du réseau à moins d'un km ayant un parking libre
124 --parametres:
    --databaseStation,in reseaustation, base de données des stations
126 --preconditions: aucune
    --postcondition: les numéros de station sont affichés
128 --exception: aucune
     procedure parkingsLibre(databaseStation:in reseauStation);
130 ------------------------------------------------------------------------

132 ------------------------------------------------------------------------
    --semantique:procedure parkingsLibre
134 --affiche les stations du réseau à moins de X metres d'une station
    --parametres:
136 --databaseStation,in reseaustation, base de données des stations
    --preconditions: aucune
138 --postcondition: les numéros des stations sont affichés
    --exception: aucune
140 procedure stationDistanteXm (databaseStation:in reseauStation);
    ------------------------------------------------------------------------
```

fonctions_principales.py

4 Structures de données nécessaires à l'application.

4.1 Création d'une station

```
1  --alternative
   demande mot de passe;
3  si incorrect
   alors
5    message de restriction;
   sinon
7    --séquence
     --s1(répétition)
9    repeter
       demande du nombre de places du parking;
11   tant que nbplace négatif
     --cs:nbplace positif
13
     --s2(répétition)
15   repeter
       demande du nombre de vélos
17   tant que nbplace<nbvelo<0
     --cs:0<=nbvelo<=nbparking
19
     --s3(séquence)
21   demande des coordonnées d'emplacement

23   --s4(alternative)
     si aucune station n'existe à cet emplacement
25   alors
       --s4.1(séquence)
27     initialisation d'une station s avec les infos fournies
       insertion en tête de cette station dans la base de donnée des
           stations
```

```
      message de reussite de l'opération
    sinon
      message d'echec de l'opération
    fin si
fin si
```

creeStation.py

4.2 Destruction d'une station

```
  --(alternative)
  demande mot de passe;
  si incorrect
  alors
     message de restriction;
  sinon si base de donnée vide
  alors
     message d'echec
  sinon
     --répétition
     repeter
       demande du numéro de la station à détruire
     tant que numéro invalide
     --cs:numéro valide
     --séquence
     enlever la station spécifiée de la base de données
     mettre à jour le réseau
     message de reussite
  fin si
```

enleveStation.py

4.3 Consultation de la base de données

```
 --(alternative)
  demande mot de passe;
  si incorrect
  alors
     message de restriction;
  sinon
    --séquence
     affichage détaillé des stations du réseau
     (coordoonées, stations à proximité ...)
     affichage détaillé des abonnés du réseau
     (identifiants,credit,caution...)
  fin si
```

infoBaseDonnee.py

4.4 Abonnement d'un utilisateur

```
  --séquence
  demande des identifiants(code postal et num)
  demande si en possession de caution de 150euros
  --alternative
  si en possession de caution et de credit suffisant
```

```
alors
   insersion en tete de l'utilisateur à la base de donnée
   message de réussite
sinon
   message d'echec
fin si
```

abonnement.py

4.5 Prise de vélo par un abonné

```
--séquence
demande des identifiants(code postal et num)
--alternative
si identifiants incorrects
alors
   message d'echec
sinon si (credit insuffisant ou abonné encore en possession d'un velo)
alors
   message d'echec
sinon
   --séquence
   demande de la station
   --alternative
   si la station dispose de vélos
   alors
     demande du numéro du vélo qu'il veut
     enlever ce vélo de la station
     l'inserer a la liste des vélos pris par l'abonné
     memoriser le temps de prise du vélo,
     (on utilise le paquetage calendar de la librairie Ada (temps=
        integer(seconds(clock))
           sinon
     message d'echec
   fin si
fin si
```

prendreVelo.py

4.6 Restitution du vélo

```
--séquence
demande d' identifiants(code postal et num)
--alternative
si identifiants incorrects
alors
   message d'echec
   (Pour éviter les pertes de vélo,
   seuls les abonnés ont le droit de restituer un vélo)
sinon
   --séquence
   demande de la station où il souhaite ranger le vélo
   --alternative
   si la station ne dispose pas de place de parking
   alors
     message d'echec
   sinon
     --séquence
```

```
        enlever le vélo de la liste des vélos pris par l'abonné
        l'inserer a de la liste des vélos de la station

        --tempsMis = temps actuel - temps de prise du vélo
        --temps actuel= integer(seconds(clock)) (librairie calendar)
        --temps de prise du velo = user.tempsprise

        débiter le compte de l'abonné,
        (en fonction du nombre de temps passé et de l'état du vélo)
    fin si
fin si

------ Remarque:
Le solde débité, vérifie la fonction partie entiere suivante
f(X)= -1/2*partEnt(-1/3 * X) - 1/2
où X représente le temps en minutes mis avec le vélo.(X>0)
et f(X) la somme en euros qui sera debitée(X>0 => f(X)>=0)
C'est l'équation de la fonction en escalier suivante:
  * longeur des marches = 30 (min)
  * hauteur entre les marches = 0.5 (euros)
```

restituerVelo.py

4.7 Consultation de son compte

```
--séquence
demande d' identifiants(code postal et num)
--alternative
si identifiants incorrects
alors
   message d'echec
sinon
   --séquence
   afficher son credit, sa caution, le numéro du vélo qu'il a pris
   demander de combien il veut créditer son compte
   ajouter la somme mentionnée à son crédit.
fin si
```

monCompte.py

4.8 Mini carte et stations avec parkings libres

Il s'agit ici d'un simple affichage à l'écran en utilisant les procédures d'affichage définies plus haut.

4.9 Stations distantes de X mètres d'une station de référence

```
    table, est de type reseauStation et va jouer un rôle d'accumulateur
    --séquence
    demander la station de référence
    (x,y)<-coordonnées de la station de référene
    deb pointe sur la premiere station du réseau
    --répétition (tant que... faire)

    tant que deb n'est pas vide faire
      (debx,deby)<-coordoonées de la station courante
      d<-distance entre station courante et station de référence
      --alternative
      si  (0<d<=X)
     alors
       memoriser la station courante en l'inserant en tete de la table
       deb pointe sur la prochaine station du réseau
     sinon
    (--d=0 donc on n'a pas besoin de mémoriser la station de référence)
      deb pointe sur la prochaine station du réseau
    fin si
    fin tant que
    --la table contient l'ensemble des stations à moins de Xm
    --de la station de référence
    --hormis la station de référence elle même

    afficher la table
```

stationDistanteXm.py

5 Interface machine de ce système.

L'interface machine du système sera présenté comme suit :

```
1
   ------------------------------Administrateur------------------------------
3
   put_line((+). Créer une station);
5  put_line((-). Detruire une station);
   put_line((*). Consulter base de données);
7  ------------------------------------------------------------------------



9
   ************************************************************
11 put_line(Chers utilisateurs, Bienvenue dans le réseau VeloTlse );
   put_line(Choisissez une option.);
13 ************************************************************

15 ------------------------------Utilisateurs--------------------------------

17 put_line(a. S'abonner à VeloToulouse);
   put_line(b. Prendre un vélo);
19 put_line(c. Restituer un vélo);
   put_line(d. Accéder à son compte);
21 ------------------------------------------------------------------------


23
   ------------------------------Stations------------------------------------
25
   put_line(e. Consulter la mini carte);
27 put_line(f. Stations distantes de moins de 1km avec parking libre);
   put_line(g. Stations à X m(à préciser) d'une station de reference);
29 put_line(q. Quitter);
   ------------------------------------------------------------------------
```

menu.py

On répètera son affichage tant que le choix de l'utilisateur ou de l'administrateur ne figure pas parmi les caractères proposés.
Une fois que le choix sera correct, on réalisera les instructions nécessaires puis on réaffichera le menu pour qu'il choisisse une autre option à sa guise.
Le système s'arrêtera lorsque le choix sera `q. Quitter`
On aura donc besoin d'un booleen nommé repeat et initialisé à `true` qui permettra de réafficher le menu tant que sa valeur n'est pas `false`. (sa valeur sera modifiée lorsque notre choix sera `q. Quitter`).

Troisième partie

Programmation de ce système en ADA.

6 Programmes de test

Le programme a été testé dans sa globalité. Voici l'ensemble des tests effectués mettant en oeuvre le bon fonctionnement du programme.

Interface machine

- Saisie d'un caractère ne figurant pas parmi ceux proposés : réaffichage du menu
- Saisie d'un caractère correct : exécution du bloc d'instructions lié à l'option choisie puis retour au menu principal.
- Saisie du caractère q pour quitter : Le programme est stoppé.

Création d'une Station

- Saisie d'un mot de passe incorrect : Affichage du message "reservé à l'administration" et retour au menu principal.
- Saisie d'un mot de passe correct :
  ⇒ *demande du nombre de places du parking :*
  - Saisie d'un caractère : Affichage du message "entrer un chiffre SVP" et retour au menu principal.
  - Saisie d'un nombre négatif : nombre de places du parking redemandée.
  - Saisie d'un nombre positif :
    ⇒ *demande du nombres de vélos de la station :*
    - Saisie d'un nombre strictement négatif ou supérieur au nombre de places du parking : nombre de vélos redemandé.
    - Saisie d'un nombre positif et inférieur au nombre de place du parking :
      ⇒ *demande des coordoonnées de la station :*
      - Saisie de coordoonées d'une station déjà créée : Affichage du message "une station existe déjà à cet emplacement" et retour au menu principal.
      - Saisie de coordonnées non utilisées : Affichage du message "opération effectuée" et retour au menu principal.

Destruction d'une Station

Si aucune station ne figure dans la base de données, le message "aucune station créee s'affiche" et le programme retourne au menu principal. Sinon,

- demande du numéro de la station à détuire :
  - Saisie d'un numéro de station inexistant : numéro redémandé
  - Saisie du numéro d'une station existante : affichage du message "opération effectuée"' et retour au menu principal

Consultation de la base de donnée

- Aucune station créée : rien ne s'affiche et retour au menu principal
- Une station est créée :affichage du numéro, du nombre de places de parking, des vélos disponobles, des numéros des stations à moins d'un kilomètre de celle-ci et des numéros de stations à proximité ayant un parking libre.

- Une station est créée puis détruite : aucune information concernant ladite station ne s'affiche. La station n'est plus prise en compte dans la liste des stations à proximité d'une autre station.

Abonnement
- Saisie d'identifiants déjà utilisés par un autre abonné : affichage du message "identifiants incorrects" et retour au menu principal
- Saisie d'un numéro non utilisé et d'un code postal(déjà utilisé ou non) :affichage du message "vous êtes abonnés" et retour au menu principal

Location d'un vélo
- Saisie d'identifiants erronés : affichage du message : "identifiants erronés" et retour au menu principal
- Saisie d'identifiants d'un abonné ayant un solde et/ou une caution insuffisants : affichage du message "votre compte n'est pas crédité." et retour au menu principal.
- Saisie d'identifiants d'un abonné ayant un compte crédité mais étant encore en possession d'un vélo : affichage du message "vous êtes encore en possession d'un vélo." et retour au menu principal
- Saisie d'identifiants d'un abonné remplissant les conditions :
  ⇒ *demande de la station où il veut prendre le vélo*
  La base de données ne contient aucune station :affichage du message "aucune station n'a été créée"
  - Saisie d'un numéro de station inexistant : numéro redémandé
  - Saisie du numéro d'une station n'ayant aucun vélo : affichage du message "Cette station ne contient aucun vélo" et retour au menu
  - Saisie du numéro d'une station ayant au moins un vélo :
    ⇒ *demande du vélo qu'il désire*
  - Saisie d'un numéro invalide : numéro redemandé
  - Saisie d'un numéro valide : affichage de l'heure de prise du vélo et du message "vélo à votre disposition"
    ⇒ *l'abonné decide s'il vole le vélo[O] ou pas[N]*
    - Saisie de O : retour au menu principal, lorsque l'abonné consulte son compte sa caution vaut 0 euro
    - Saisie de N : retour au menu principal, le compte de l'abonné n'est pas affecté

Restitution d'un vélo
- Saisie d'identifiants d'un abonné n'ayant aucun vélo en sa possession : affichage du message : "vous n'avez aucun vélo à rendre" et retour au menu principal
- Saisie d'identifiants d'un abonné en possession d'un vélo :
  ⇒ *demande de la station où il veut restituer le vélo*
  - Saisie d'un numéro d'une station ne disposant pas de place de parking pour ranger le vélo :affichage du message "pas de place" et retour au menu principal
  - Saisie du numéro d'une station disposant de places : affichage du nombre de temps passé avec le vélo, et du message "Merci de l'avoir rendu".
    Si le temps est inférieur à 30 minutes, crédit identique lorsqu'on consulte le compte de l'abonné.
    Si le temps est dans l'intervalle [31 ,60] minutes le crédit a diminué de 50centimes.
    etc $\cdots$

Consultation de compte

- abonné ayant volé un vélo : la caution est à 0 euro
- abonné ayant passé moins de 30 minutes avec un vélo : le solde est inchangé
- abonné ayant passé plus de 30 minutes avec un vélo : le solde a été débité en fonction du nombre de temps
- possibilité de créditer son compte.

mini carte Seules les stations n'ayant pas été détuites sont affichées avec leur coordoonées respectives.

Stations à moins d'un kilomètre d'une station donnée
- La base de donnée n'a qu'une station : rien n'est affiché
- La base de donnee a des stations distantes à plus d'un km et d'autres distantes à moins d'un km : seules les stations distantes à moins d'un km de la station de référence qu'on aura au préalable choisi sont affichés.
- On a trois stations à moins d'un km les unes des autres, puis on supprime une station : la station supprimée ne figure plus dans la liste des stations distantes à moins d'un km des deux stations restantes.

Conclusion sur les tests

Les tests effectués assurent le bon fonctionnement de mon programme dans sa globalité. Toutes les exceptions sont traitées sans que le programme ne s'interrompe. Ce qui permet de conserver les données enregistrées tout au long du programme. Le seul moyen d'interrompre le processuss (à moins d'éteindre son ordinateur ou de fermer le terminal) est de quitter à partir du menu principal en appuyant sur le caractère q.

7 Quelques Informations

Vous trouverez ci-joint les fichiers sources suivants :

1. `libansi` il s'occupe de l'effaçage d'écran et de la coloration du texte affiché en vue de la convivialité
2. Les paquetages suivant :
   - `p_velo`
   - `p_utilisateur`
   - `p_station`
3. Mon programme principal nommé `velotlse.adb`.
4. Un paquetage nommé `p_menu` comportant les principales fonctions mentionnées plus haut. Je l'ai créé pour des raisons d'aération et de clarté du programme principale.

Les sources on été imprimées via la commande `a2ps` à partir d'un terminal sous Linux. l'abbrévation `tmib` de `Printed by tmib` représente les premières lettres de mon nom Tchomgue Mieguem Ivan Brunel.

8 Conclusion

Le sujet permet une mise en œuvre pratique des connaissances reçues en cours. La méthode des raffinages retrouve dans ce cadre toute son importance pour une bonne conception de cette application. L'utilisation des pointeurs permet une gestion optimale de la base de données et un stockage 'infini' de ces données.

Concernant le choix du langage, je trouve que le choix de programmer de façon impérative, est bien adapté pour ce sujet. Bien que le langage ne nous permette pas réellement de scinder les interfaces administrateur et utilisateur.

A Listings

```
package libansi is

        --deplace le curseur de x espacements vers la droite
        --et de y vers le bas
        procedure cursor_moveto(x: integer; y:integer);

        --efface l'ecran
        procedure clear_screen;

        --change la coloration du curseur
        procedure cursor_color(x: integer);

end libansi;
```

```
with ada.text_io, ada.strings.fixed;
use ada.text_io, ada.strings.fixed;
with ada.text_io; use ada.text_io;

with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io; use ada.float_text_io;

package body libansi is
    CSI : constant string := Character'val(27)&"[";
    ESC: CONSTANT Character := Character'Val(27);


    procedure cursor_moveto(x: integer; y:integer) is
    begin
        put(CSI&Trim(integer'image(y),ada.strings.left)&";"&Trim(integer'image(x
),ada.strings.left)&"H");
    end cursor_moveto;


    procedure clear_screen is
    begin
        put(CSI&"2J");
    end clear_screen;


    procedure cursor_color(x: integer) is
    begin
        if (x<0) or (x>7) then
            return;
        else
            put(CSI&Trim(Integer'image(30+x), ada.strings.left)&"m");
        end if;
    end cursor_color;


end libansi;
```

```
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io; use ada.float_text_io;

package p_velo is

    type listeVelo is private;
    exception_velo:exception;

    ------------------------------------------------------------------------
    --semantique:fonction listevide creer une liste vide
    --parametres: aucun
    --type-retour: listeVelo
    --pre-condition: aucune
    --post-condition: est_vide (l) vaut vrai
    --exception: aucune
    function listevide return listeVelo;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: fonction est_vide teste si une liste l est vide
    --parametres: l donnee type listeVelo
    --type retour: booleen
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    function est_vide (l:in listeVelo) return boolean;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: enlever un element e de la liste l (liste vide ou non vide)
    --parametres: l donnee/resultat type listeVelo
    --e donnee type entier
    --pre-condition: aucune
    --post-condition: e n'appartient pas a la liste
    --exception: aucune
    procedure enlever(l:in out listeVelo; e:in integer);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: procedure inserer_en_tete
    --insere en tete de la listeNumstation l l'elt nouveau
    --parametres: l donnee/resultat type listeVelo
    --nouveau donnee type entier
    --pre-condition: aucune
    --post-condition: nouveau appartient a la liste
    --exception: aucune
    procedure inserer_en_tete(l:in out listeVelo; nouveau:in integer);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure genereVelos
    --cree une liste de velos pour un parking
    --parametres:
    --nbrevelo,in integer,nombre de velos a generer
    --databaseVelo, in out listeVelo, base de donnees des velos
    --veloGenere,out listeVelo,la liste des velos generes
    --preconditions:
```



```
    --aucune
    --postcondition:
    --la base de donnee des velos a nbreVelo en plus.
    --exception:aucune
    procedure genereVelos (nbreVelo:in integer;databaseVelo:in out listeVelo;
        velosGenere:out listeVelo);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
-
    --semantique: fonction rechercher recherche si e appartient a la liste l
    --retourne son adresse
    --ou null si la liste est vide ou si e n'appartient pas a la liste
    --parametres: l donnee type listeVelo
    --e in integer
    --type-retour: listeVelo
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    function rechercher(l:in listeVelo; e:in integer) return
    listeVelo;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure afficher affiche les elements de la liste l
    --parametres: l donnee type listeVelo
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    procedure afficher(l:in listeVelo);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique:fonction longueur,calcule la taille d'une liste
    --parametre:
    --l:in listeVelo
    --precondition aucune
    --postcondition aucune
    --type de retour integer
    --exception:aucune
    function longueur(l:in listeVelo) return integer;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --recupVelo recpere le numero d'un velo
    --parametre:l in listeVelo, une liste de velo
    --type de retour: in integer
    --precondition:aucune
    --postcondition:numero de velo recupere
    --exception l est vide
    function recupVelo (l:in listeVelo)return integer;
    ------------------------------------------------------------------------

    private

    type noeud;
    type listeVelo is access noeud;
    type noeud is record
```

```
        val:integer;
        suivant:listeVelo;
    end record;
end p_velo;
```

```
with libansi;use libansi;
with p_velo;use p_velo;
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io; use ada.float_text_io;
with Ada.Numerics.Elementary_Functions; use  Ada.Numerics.Elementary_Functions;

package p_station is

   type station is private;
   type reseauStation is private;
   type coordonnees is private;
   exception_reseau:exception;

   ------------------------------------------------------------------------
   --semantique:fonction listevide creer une liste vide
   --parametres: aucun
   --type-retour: reseauStation
   --pre-condition: aucune
   --post-condition: est_vide (l) vaut vrai
   --exception: aucune
   function listevide return reseauStation;
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique: fonction est_vide teste si une liste l est vide
   --parametres: l donnee type reseauStation
   --type retour: booleen
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   function est_vide (l:in reseauStation) return boolean;
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique: procedure inserer_en_tete
   --insere en tete de la reseauStation l l'elt nouveau
   --parametres: l donnee/resultat type reseauStation
   --nouveau donnee type station
   --pre-condition: aucune
   --post-condition: nouveau appartient a la liste
   --exception: aucune
   procedure inserer_en_tete(l:in out reseauStation; nouveau:in station);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique: fonction rechercher recherche si e appartient a la liste l
   --retourne son adresse
   --ou null si la liste est vide ou si e n'appartient pas a la liste
   --parametres: l donnee type reseauStation
   --e donnee type entier
   --type-retour: reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   function rechercher(l:in reseauStation; e:in integer) return
   reseauStation;
   ------------------------------------------------------------------------
   --
```



```
   --
   ------------------------------------------------------------------------
   --semantique: fonction rechercher recherche si (x,y) appartient a la liste
   --retourne son adresse
   --ou null si la liste est vide ou si e n'appartient pas a la liste
   --parametres: l donnee type reseauStation
   --x,y in float,coordonnee recherchees
   --type-retour: reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   function rechercher(l:in reseauStation; x:in float;y:in float) return
   reseauStation;
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure afficher affiche les elements de la liste l
   --parametres: l donnee type reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   procedure afficher(l:in reseauStation);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure affichage affiche les elements de la liste l
   --parametres: r,in reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   procedure affichage(r:in reseauStation);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:fonction recupere
   --retourne le champ d'un enregistrement
   --parametres:
   --s,in reseauStation,station dont on retournera le champ
   --precondition:aucune
   --postcondition:on dispose du champ souhaite
   --exception:aucune
   function recupere (s:reseauStation) return listeVelo;
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:fonction recupere
   --retourne le champ d'un enregistrement
   --parametres:
   --s,in reseauStation,station dont on retournera le champ
   --precondition:aucune
   --postcondition:on dispose du champ souhaite
   --exception:aucune
   --type de retour:integer
   function recupere (s:in reseauStation) return integer;
   function recupVal (s:in reseauStation)return station;
   function recupNum(s:in reseauStation)return integer;
   ------------------------------------------------------------------------
   --
```

```
   ------------------------------------------------------------------------
   --procedure set
   --met a jour un champ
   --parametres
   --l,in reseauStation
   --le deuxieme parametre(in) est la valeur avec laquelle on souhaite
   --mettre a jour le champ, il est de meme type que le champ
   --precondition aucune
   --postcondition aucune
   --exception:l est null
   ------------------------------------------------------------------------
   procedure set (l:in reseauStation;velolist:in listeVelo);

   procedure set(l:in reseauStation;bool:in boolean);

   procedure setStations(databaseStation:in reseauStation;x:in float);
  ------------------------------------------------------------------------
   --
   --
  ------------------------------------------------------------------------
   --procedure set<<nomduchamp>>
   --met a jour le champ <<nomduchamp>>
   --parametres
   --s,in out station
   --le(s) autres parametres (in) representent
   --la valeur avec laquelle on souhaite mettre a jour
   --precondition aucune
   --postcondition aucune
   --exception:aucune
   ------------------------------------------------------------------------
   procedure setNum(s:in out station;x:in  integer);

   procedure setPlaceLibre (s:in out station; bool:in boolean);

   procedure setNbVelos(s:in out station; x:in integer);

   procedure setVelosDispo(s:in out station; listVelos:in listeVelo);

   procedure setCoordo(s:in out station; x:in float;y:in float);
  ------------------------------------------------------------------------
  --
  --
  ------------------------------------------------------------------------
   --fonction recup<<nomduchamp>>
   --recupere le champ <<nomduchamp>>
   --parametres
   --s,in station
   --precondition aucune
   --postcondition aucune
   --type de retour:le type du champ recupere
   --exception:aucune
   ------------------------------------------------------------------------
   function recupNum(s:in station) return integer;

   function recupVelosDispo (s:in station) return listeVelo;

   function recupStations_1_km (s:in station) return reseauStation;

   function recupNbVelos(s:in station) return integer;
   ------------------------------------------------------------------------
   --
   --
```



```
   ------------------------------------------------------------------------
   --semantique: fonction stationDistanteX
   --retourne l'ensemble des stations distantes de moins de Xm
   --d'une station result donnee hormis la station result elle meme
   --parametres:
   --databaseStation,in reseauStation, la base de donnees des
   --stations
   --X,in float,la distance souhaitee
   --result:in station,la station de reference
   --pre-condition:aucune
   --post-condition: les stations distantes de moins de Xm sont retournees
   --type de retour: reseauStation
   --exception:aucune
   --tests:
   --la distante X vaut 0, aucune station n'est retournee
   --la base de donnee r est vide, on retourne null
   function stationDistanteX (databaseStation:in reseauStation;
   X:in float;ref:in station) return reseauStation;
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure afficheStationLibre
   --affiche les stations du reseau r ayant une place de parking libre
   --parametre:
   --r,in reseauStation
   --precondition:aucune
   --postcondition:les numeros de stations sont affiches
   --exception:aucune
   --tests:
   -- r est vide
   --r ne contient aucune station libre
   --r a au moins une station libre
   procedure afficheStationLibre(r:in reseauStation);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique: enlever un element e de la liste l (liste vide ou non vide)
   --parametres: l donnee/resultat type listeVelo
   --e donnee type integer
   --pre-condition: aucune
   --post-condition: e n'appartient pas a la liste
   --exception: aucune
   procedure enlever(l:in out reseauStation; e:in integer);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure statStation
   --donne un compte rendu concernant chacune des stations du reseau
   --coordonnees, velos,le numero etc
   --parametre:
   --r:in reseauStation,la base de donnee des stations
   --precondition:aucune
   --postconditon:le compte rendu de chaque station est affiche
   --exeption:aucune
   --tests:
   --la base de donnee est vide
   --la base de donnee a au moins une station
   procedure statStation (databaseStation:in reseauStation);
   ------------------------------------------------------------------------
```

```
   private
   type coordonnees is record
      absc:float;
      ordo:float;
   end record;

   type station is record
      num:integer;
      nbVelos:integer;
      coordo:coordonnees;
      placeLibre:boolean;
      VelosDispo:listeVelo;
      stations_1_km:reseauStation;
   end record;

   type noeud2;
   type reseauStation is access noeud2;
   type noeud2 is record
      val:station;
      suivant:reseauStation;
   end record;
end p_station;
```

```
with libansi; use libansi;
with p_velo;use p_velo;
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io; use ada.float_text_io;

package p_utilisateur is
    cautionInitiale: constant integer  := 150;
    creditInitial: constant float := 3.0;

    type utilisateur is private;
    type listeUtilisateur is private;
    exception_compte:exception;


    ------------------------------------------------------------------------
    --semantique:fonction listevide creer une liste vide
    --parametres: aucun
    --type-retour: listeUtilisateur
    --pre-condition: aucune
    --post-condition: est_vide (l) vaut vrai
    --exception: aucune
    function listevide return listeUtilisateur;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: fonction est_vide teste si une liste l est vide
    --parametres: l donnee type listeUtilisateur
    --type retour: booleen
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    function est_vide (l:in listeUtilisateur) return boolean;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: procedure inserer_en_tete
    --insere en tete de la listeNumstation l l'elt nouveau
    --parametres: l donnee/resultat type listeUtilisateur
    --nouveau donnee type utilisateur
    --pre-condition: aucune
    --post-condition: nouveau appartient a la liste
    --exception: aucune
    procedure inserer_en_tete(l:in out listeUtilisateur;nouveau:in utilisateur);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: fonction rechercher recherche si e appartient a laliste l
    --retourne son adresse
    --ou null si la liste est vide ou si e n'appartient pas a la liste
    --parametres: l donnee type listeUtilisateur
    --e donnee type entier
    --precision:in string, precision du champ de recherche(num,codePostal...)
    --type-retour: listeUtilisateur
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    function rechercher(l:in listeUtilisateur; e:in integer;precision:in string) r
eturn listeUtilisateur;
```



```
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique: fonction statUser
    --donne un compte rendu de chacun des abonnes
    --parametres:
    --databaseUser,in listeUtilisateur, base de donnees utilisateurs
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    --tests:
    --la base de donnee est vide
    --la base de donnee a au moins un utilisateur
    procedure statUser(databaseUser:in listeUtilisateur);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --fonction recup<<nomduchamp>>
    --recupere le champ <<nomduchamp>>
    --parametres
    --userlist,in listeUtilisateur
    --precondition aucune
    --postcondition aucune
    --type de retour:le type du champ recupere
    --exception:userlist est vide
    ------------------------------------------------------------------------
    function recupCredit(userlist:in listeUtilisateur) return float;
    function recupCaution(userlist:in listeUtilisateur) return integer;
    function recupVeloPris(userlist:in listeUtilisateur) return listeVelo;
    function recuptempsprise(userlist:in listeUtilisateur) return integer;
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --procedure set<<nomduchamp>>
    --met a jour le champ <<nomduchamp>>
    --parametres
    --s,in listeUtilisateur
    --le(s) autres parametres (in) representent
    --la valeur avec laquelle on souhaite mettre a jour
    --precondition aucune
    --postcondition aucune
    --exception:userlist est vide
    ------------------------------------------------------------------------
    procedure setCredit(userlist:in listeUtilisateur;x:in float);
    procedure setCaution (userlist:in listeUtilisateur;x:in integer);
    procedure setVeloPris(userlist:in listeUtilisateur;velo_list:in listeVelo);
    procedure settempsprise(userlist:in listeUtilisateur;temps:in integer);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --procedure setAll
    --met a jour tous les champs d'un utilisateur
    --parametres
    --user,in out utilisateur
    --le(s) autres parametres (in) representent
    --la valeur avec laquelle on souhaite mettre a jour
    --precondition aucune
    --postcondition aucune
```

```
    --exception:aucune
    procedure setAll(user:in out utilisateur;num:in integer;BP:in integer;
        credit:in float;caution:in integer;velos:in listeVelo;temps:in integer);



    private
    type utilisateur is record
        num:integer;
        codePostal:integer;
        credit:float;
        caution:integer;
        veloPris:listeVelo;
        tempsprise:integer;--temps de prise du velo en secondes
    end record;

    type noeud;
    type listeUtilisateur is access noeud;
    type noeud is record
        val:utilisateur;
        suivant:listeUtilisateur;
    end record;
end p_utilisateur;
```

```
with libansi;use libansi;
with ada.calendar;use ada.calendar;
with p_velo; use p_velo;
with p_station;use p_station;
with p_utilisateur;use p_utilisateur;
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io; use ada.float_text_io;
with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
package p_menu is

   ------------------------------------------------------------------------
   --Semantique: fonction qui affichage le menu de faÃ§on conviviale et fiable
   --puis recupere le choix entre au clavier.
   --parametres:aucun
   --exception:aucune
   --type de retour:caractere
   function choixMenu return character;
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique: procedure creeStation
   --cree une nouvelle station
   --parametres:
   --databaseStation,in out reseauStation,base de donnee des stations
   --databaseVelo,in out listeVelo,base de donnees des velos deja crees
   --les numeros des velos sont generes automatiquement
   --chaque velo a un numero unique
   --on a besoin de cette base de donnees pour eviter les repetitions
   --preconditions:
   --le numero de la station est unique
   --postcondition:
   --le reseau a une station de plus
   --exception:aucune
   procedure creeStation(databaseStation:in out reseauStation;
       databaseVelo:in out listeVelo);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure enleveStation
   --enleve une station du reseau sans perube son fonctionnement
   --parametres:databaseStation,in reseauStation,base de donnee
   --pre:reseau non vide
   --post:le reseau a une station en moins
   --exception:aucune
   procedure enleveStation(databaseStation:in out reseauStation);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --procedure consulteBaseDedonnees
   --donne un compte rendu des bases de donnees
   --parametres:
   --r,in reseauStation,base de donnee des stations
   --lu:in listeUtilisateur,base de donnee des utilisateurs
   procedure consulteBaseDedonnees(r:reseauStation;lu:listeUtilisateur);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
```



```
   --semantique:procedure abonnement
   --ajoute un utilisateur a la base de donnees utilisateurs.
   --parametres:
   --databaseUser,in out listeUtilisateur, base de donnees
   --preconditions:
   --avoir un numero valide (on suppose qu'un numero est valide s'il n'a pas
   --encore ete utilise,donc ne figure pas dans la base de donnees)
   --avoir une caution de 150 euros et un credit de 3 euros
   --postcondition: un utilisateur est ajoute a la base de donnees.
   --exception: aucune
   procedure abonnement (databaseUser:in out listeUtilisateur);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure prendreVelo
   --autorise un abonne a prendre un velo
   --parametres:
   --databaseVelo,in out listeVelo, base de donnees des velos
   --databaseUser,in listeUtilisateur, base de donnees des utilisateurs
   --databaseStation:in reseauStation,base de donnees des stations
   --preconditions:
   --avoir un numero valide (on suppose qu'un numero est valide s'il
   --figure pas dans la base de donnees)
   --avoir un credit d'au moins 3 euros
   --velo disponible dans la station
   --postcondition: un capteur signale l'absence du velo pris
   --dans la base de donnees de velos et
   --le numero de velo s'ajoute a la liste de velos pris par l'utilisateur
   --exception: aucune
   procedure prendreVelo(databaseVelo:in out listeVelo;
       databaseUser:in listeUtilisateur;databaseStation:in
       reseauStation);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique procedure restituerVelo
   --restitue un velo a une station
   --parametres:
   --databaseVelo,in out listeVelo, base de donnees des velos
   --databaseUser,in listeUtilisateur, base de donnees des utilisateurs
   --databaseStation:in reseauStation,base de donnees des stations
   --preconditions:
   --la station dispose d'au moins une place libre
   --postcondition:
   --un capteur signale qu'un velo a ete restituer
   --dans la base de donnees des velos
   --le credit est debite en fonction du temps passe avec le velo
   --exception:aucune.
   procedure restituerVelo(databaseVelo:in out listeVelo;
       databaseUser:in listeUtilisateur; databaseStation:in reseauStation);
   ------------------------------------------------------------------------
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure monCompte
   --autorise un abonne a consulter les details de son compte
   --parametres:
   --databaseUser,in listeUtilisateur, base de donnees des utilisateurs
   --preconditions:
   --avoir un numero valide (on suppose qu'un numero est valide s'il
```

```
    --figure pas dans la base de donnees)
    --postcondition: les infos sont affiches
    --exception: aucune
    procedure monCompte(databaseUser:in listeUtilisateur);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------
    --semantique:procedure miniCarte affiche les stations
    --du reseau a leur emplacement (en fonction de leurs coordoonees
    --parametres:databaseStation:in reseauStation,la base de donnee
    --precondition:aucune
    --postcondition:stations affichees
    --exception:aucune
    procedure miniCarte (databaseStation:in reseauStation);
    ------------------------------------------------------------------------
    --
    --
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure stationDistanteXm
    --affiche l'ensemble des stations distantes de moins de Xm
    --parametres: databaseStation:in reseauStation,la base de donnees
    --precondition:la base de donnee n'est pas vide
    --postcondition:les stations sont affichees
    --exception:aucune
    procedure stationDistanteXm (databaseStation:in reseauStation);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure parkingsLibre
    --affiche les stations ayant une place de parking libre
    --parametres:databaseStation,in reseauStation,base de donnee
    --pre:base de donnee non vide
    --post:stations affichees
    --exception:aucune
    procedure parkingsLibre(databaseStation:in reseauStation);
    ------------------------------------------------------------------------
    --
    --
    ------------------------------------------------------------------
    --procedure quitter
    --quitte le menu en laissant un message et en arretant la boucle
    --qui reaffiche le menu
    --parametre:bool,in boolean, la condition d'arret de la boucle
    --pre,post,exception:aucun
    procedure quitter (bool:in out boolean);
    ------------------------------------------------------------------
    --
    --

end p_menu;
```

```
package body p_velo is

   ------------------------------------------------------------------------
   --semantique:fonction listevide creer une liste vide
   --parametres: aucun
   --type-retour: listeVelo
   --pre-condition: aucune
   --post-condition: est_vide (l) vaut vrai
   --exception: aucune
   ------------------------------------------------------------------------
   function listevide return listeVelo is
   begin
      return null;
   end listevide;
   --
   --
   ------------------------------------------------------------------------
   --semantique: fonction est_vide teste si une liste l est vide
   --parametres: l donnee type listeVelo
   --type retour: booleen
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   ------------------------------------------------------------------------
   function est_vide (l:in listeVelo) return boolean is
   begin
      return (l=null);
   end est_vide;
   --
   --
   ------------------------------------------------------------------------
   --semantique: enlever un element e de la liste l (liste vide ou non vide)
   --parametres: l donnee/resultat type listeVelo
   --e donnee type entier
   --pre-condition: aucune
   --post-condition: e n'appartient pas a la liste
   --exception: aucune
   ------------------------------------------------------------------------
   procedure enlever(l:in out listeVelo; e:in integer) is
      p:listeVelo;
      q:listeVelo;
   begin
      if (l/=null and then l.all.val=e)
      then
         l:=l.all.suivant;
      elsif l=null
      then
         null;
      else
         p:=l;
         q:=l;
         while (p/=null and then p.all.val/=e) loop
            q:=p;
            p:=p.all.suivant;
         end loop;
         --p=null ou p.all.val=e
         if p=null
         then
            null;
         else
            q.all.suivant:=p.all.suivant;
         end if;
```



```
      end if;
   end enlever;
   --
   --
   ------------------------------------------------------------------------
   --semantique: procedure inserer_en_tete
   --insere en tete de la listeNumstation l l'elt nouveau
   --parametres: l donnee/resultat type listeVelo
   --nouveau donnee type entier
   --pre-condition: aucune
   --post-condition: nouveau appartient a la liste
   --exception: aucune
   ------------------------------------------------------------------------
   procedure inserer_en_tete(l:in out listeVelo; nouveau:in integer) is
      pins:listeVelo; --pointeur d'insertion
   begin
      pins:= new noeud'(nouveau,l);
      l:=pins;
   end inserer_en_tete;
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure genereVelos
   --cree une liste de velos pour un parking
   --parametres:
   --nbrevelo,in integer,nombre de velos a generer
   --databaseVelo, in out listeVelo, base de donnees des velos
   --velosGenere,out listeVelo,la liste des velos generes
   --preconditions:
   --aucune
   --postcondition:
   --la base de donnee des velos a nbreVelo en plus.
   --nbreVelos ont ete generes
   --exception:aucune
   ------------------------------------------------------------------------
   procedure genereVelos (nbreVelo:in integer;databaseVelo:in out listeVelo;
      velosGenere:out listeVelo) is
      v:integer;        --v est un velo identifie par son numero
   begin
      for i in 1..nbreVelo loop
         if databaseVelo=null
         then
            v:=1;
         else
            v:=databaseVelo.all.val+1;
         end if;
         inserer_en_tete(databaseVelo,v);
         inserer_en_tete(velosGenere,v);
      end loop;
   end genereVelos;
   --
   --
   ------------------------------------------------------------------------
   --semantique: fonction rechercher recherche si e appartient a la liste l
   --retourne son adresse
   --ou null si la liste est vide ou si e n'appartient pas a la liste
   --parametres: l donnee type listeVelo
   --e in integer
   --type-retour: listeVelo
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
```

```
   ------------------------------------------------------------------------
   function rechercher(l:in listeVelo; e:in integer) return
      listeVelo is
      courant:listeVelo;
   begin
      courant:=l;
      while (courant/=null and then courant.all.val/= e) loop
         courant:=courant.all.suivant;
      end loop;
      --courant.all.val=e ou courant=null
      return courant;
   end rechercher;
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure afficher_liste afficher les elements de la liste l
   --parametres: l donnee type listeVelo
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   ------------------------------------------------------------------------
   procedure afficher(l:in listeVelo) is
   begin

      if (l=null)
      then
         put ("]");
      else
         put (l.all.val,3);
         put("->");
         afficher(l.all.suivant);
      end if;
   end afficher;
   --
   --
   ------------------------------------------------------------------------
   --semantique:fonction length,calcule la taille d'une liste
   --parametre:
   --l:in listeVelo
   --precondition aucune
   --postcondition aucune
   --type de retour integer
   --exception:aucune
   ------------------------------------------------------------------------
   function longueur(l:in listeVelo) return integer is
      p:listeVelo;--pointeur de parcours
      cmpt:integer;--compteur
   begin
      p:=l;
      cmpt:=0;
      while(p/=null) loop
         p:=p.all.suivant;
         cmpt:=cmpt+1;
      end loop;
      --cmpt est la taille de la liste, p=null
      return cmpt;
   end longueur;
   --
   --
   ------------------------------------------------------------------------
    --recupVelo recpere le numero d'un velo
    --parametre:l in listeVelo, une liste de velo
```



```
    --type de retour: in integer
    --precondition:aucune
    --postcondition:numero de velo recupere
    --exception l est vide
   ------------------------------------------------------------------------
   function recupVelo(l:in listeVelo) return integer is
   begin
      if l=null
      then
        raise exception_velo;
      else
        return l.all.val;
      end if;
   end recupVelo;
   ------------------------------------------------------------------------


end p_velo;
```

```
package body p_station is

   ------------------------------------------------------------------------
   --semantique:fonction listevide creer une liste vide
   --parametres: aucun
   --type-retour: reseauStation
   --pre-condition: aucune
   --post-condition: est_vide (l) vaut vrai
   --exception: aucune
   ------------------------------------------------------------------------
   function listevide return reseauStation is
   begin
      return null;
   end listevide;
   --
   --
   ------------------------------------------------------------------------
   --semantique: fonction est_vide teste si une liste l est vide
   --parametres: l donnee type reseauStation
   --type retour: booleen
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   ------------------------------------------------------------------------
   function est_vide (l:in reseauStation) return boolean is
   begin
      return (l=null);
   end est_vide;
   --
   --
   ------------------------------------------------------------------------
   --semantique: procedure inserer_en_tete
   --insere en tÃªte de la reseauStation l l'elt nouveau
   --parametres: l donnee/resultat type liste
   --nouveau donnee type station
   --pre-condition: aucune
   --post-condition: nouveau appartient a la liste
   --exception: aucune
   ------------------------------------------------------------------------
   procedure inserer_en_tete(l:in out reseauStation; nouveau:in station) is
      pins:reseauStation; --pointeur d'insertion
   begin
      pins:= new noeud2'(nouveau,l);
      l:=pins;
   end inserer_en_tete;

   ------------------------------------------------------------------------
   --semantique: fonction rechercher recherche si e appartient a la liste l
   --retourne son adresse
   --ou null si la liste est vide ou si e n'appartient pas a la liste
   --parametres: l donnee type listeUtilisateur
   --e donnee type entier
   --type-retour: reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   ------------------------------------------------------------------------
   function rechercher(l:in reseauStation; e:in integer) return
      reseauStation is
      courant:reseauStation;
   begin
      courant:=l;
```



```
      while (courant/=null and then courant.all.val.num /= e) loop
         courant:=courant.all.suivant;
      end loop;
      --courant.all.val.num=e ou courant=null
      return courant;
   end rechercher;
   --
   --
   ------------------------------------------------------------------------
   --semantique: fonction rechercher recherche si (x,y) appartient a la liste
   --retourne son adresse
   --ou null si la liste est vide ou si e n'appartient pas a la liste
   --parametres: l donnee type reseauStation
   --x,y,in float,coordoonees qu'on cherche
   --type-retour: reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   ------------------------------------------------------------------------
   function rechercher(l:in reseauStation; x:in float;y:in float) return
      reseauStation is
      courant:reseauStation;
   begin
      courant:=l;
      while (courant/=null and then (courant.all.val.coordo.absc /= x
        or courant.all.val.coordo.ordo /= y)) loop

         courant:=courant.all.suivant;
      end loop;
      --courant.all.val.coordo=(x,y) ou courant=null
      return courant;
   end rechercher;
   --
   --
   ------------------------------------------------------------------------
   --semantique:procedure afficher_liste afficher les elements de la liste l
   --parametres: l donnee type reseauStation
   --pre-condition: aucune
   --post-condition: aucune
   --exception: aucune
   ------------------------------------------------------------------------
-
   procedure afficher(l:in reseauStation) is
   begin

      if (l=null)
      then
         put ("]");
      else
         put (l.all.val.num,3);
         put("->");
         afficher(l.all.suivant);
      end if;
   end afficher;
   --
   --
   ------------------------------------------------------------------------
   --semantique:affichage personnalise
   --affiche convivialement
   --parametres:
   --r,in reseauStation
   --pre:aucune
```

```
   --post:affichage
   --exception:aucune
   ------------------------------------------------------------------------
   procedure affichage (r:in reseauStation) is
      x,y:float;
   begin
      if r=null
      then
         null;
      else
         x:=r.all.val.coordo.absc;
         y:=r.all.val.coordo.ordo;

         cursor_moveto(integer(x),integer(y));
         cursor_color(3);
         put("<");
         put("ID");
         put(r.all.val.num,1);
         put(":(");
         put(x,0,0,0);
         put(',');
         put(y,0,0,0);
         put(")>");
         cursor_moveto(0,0);
         affichage(r.all.suivant);
      end if;
      cursor_color(7);
   end affichage;
   --
   --
   ------------------------------------------------------------------------
   --semantique:fonction recupere
   --retourne le champ d'un enregistrement
   --parametres:
   --s,in reseauStation,station dont on retournera le champ
   --precondition:aucune
   --postcondition:on dispose du champ souhaite
   --exception:s est vide
   --type de retour:listeVelo
   ------------------------------------------------------------------------
   function recupere (s:reseauStation) return listeVelo is
   begin
      if s=null
      then
         raise exception_reseau;
      else
         return s.all.val.velosDispo;
      end if;
   end recupere;
   --
   --
  ------------------------------------------------------------------------
   --semantique:fonction recupVal
   --retourne le champ d'un enregistrement
   --parametres:
   --s,in reseauStation,reseau dont on retournera la station
   --precondition:aucune
   --postcondition:on dispose du champ souhaite
   --exception:s est vide
   --type de retour:station
   ------------------------------------------------------------------------
   function recupVal(s:in reseauStation) return station is
```



```
   begin
      if s=null
      then
        raise exception_reseau;
      else
        return s.all.val;
      end if;
   end recupVal;
   --
   --
   ------------------------------------------------------------------------
   --semantique:fonction recupere
   --retourne le champ d'un enregistrement
   --parametres:
   --s,in reseauStation,station dont on retournera le champ
   --precondition:aucune
   --postcondition:on dispose du champ souhaite
   --exception:s est vide
   --type de retour:integer
   ------------------------------------------------------------------------
   function recupere (s:in reseauStation) return integer is
   begin
      if s=null
      then
         raise exception_reseau;
      else
      return s.all.val.nbVelos;
      end if;
   end recupere;
function recupNum (s:in reseauStation) return integer is
   begin
      if s=null
      then
         raise exception_reseau;
      else
         return s.all.val.num;
      end if;
   end recupNum;
   --
   --
   ------------------------------------------------------------------------
   --procedure set
   --met a jour un champ
   --parametres
   --l,in reseauStation
   --le deuxieme parametre(in) est la valeur avec laquelle on souhaite
   --mettre a jour le champ, il est de meme type que le champ
   --precondition aucune
   --postcondition aucune
   --exception:l est null
   ------------------------------------------------------------------------
   procedure set (l:in reseauStation;velolist:in listeVelo) is
   begin
      if l=null
      then
        raise exception_reseau;
      else
        l.all.val.velosDispo:=velolist;
      end if;
   end set;

   procedure set(l:in reseauStation;bool:in boolean) is
```

```
   begin
      if l=null
      then
        raise exception_reseau;
      else
        l.all.val.placeLibre:=bool;
      end if;
   end set;

   procedure setStations(databaseStation:in reseauStation;x:in float) is
      temp:reseauStation;
   begin
      temp:=databaseStation;
      while temp/=null loop
         temp.all.val.stations_1_Km:=
            stationDistanteX(databaseStation,x,temp.all.val);
         temp:=temp.all.suivant;
      end loop;
      --temp=null
   end setStations;
-------------------------------------------------------------------------
   --
   --
-------------------------------------------------------------------------
   --procedure set<<nomduchamp>>
   --met a jour le champ <<nomduchamp>>
   --parametres
   --s,in out station
   --le(s) autres parametres (in) representent
   --la valeur avec laquelle on souhaite mettre a jour
   --precondition aucune
   --postcondition aucune
   --exception:aucune
   -------------------------------------------------------------------------
   procedure setNum(s:in out station;x:in  integer)is
   begin
      s.num:=x;
   end setNum;

   procedure setPlaceLibre (s:in out station; bool:in boolean) is
   begin
      s.placeLibre:=bool;
   end setPlaceLibre;

   procedure setNbVelos(s:in out station; x:in integer) is
   begin
      s.nbVelos:=x;
   end setNbVelos;

   procedure setVelosDispo(s:in out station; listVelos:in listeVelo) is
   begin
      s.VelosDispo:=listVelos;
   end setVelosDispo;

   procedure setCoordo(s:in out station; x:in float;y:in float) is
   begin
      s.coordo.absc:=x;
      s.coordo.ordo:=y;
   end setCoordo;
-------------------------------------------------------------------------
   --
   --
```



```
-------------------------------------------------------------------------
   --fonction recup<<nomduchamp>>
   --recupere le champ <<nomduchamp>>
   --parametres
   --s,in station
   --precondition aucune
   --postcondition aucune
   --type de retour:le type du champ recupere
   --exception:aucune
   -------------------------------------------------------------------------
   function recupNum(s:in station) return integer is
   begin
      return s.num;
   end recupNum;

   function recupVelosDispo (s:in station) return listeVelo is
   begin
      return s.VelosDispo;
   end recupVelosDispo;

   function recupStations_1_km (s:in station) return reseauStation is
   begin
      return s.stations_1_km;
   end recupStations_1_km;

   function recupNbVelos(s:in station) return integer is
   begin
      return s.nbVelos;
   end recupNbVelos;
   -------------------------------------------------------------------------
   --
   --
   -------------------------------------------------------------------------
   --semantique: fonction stationDistanteX
   --retourne l'ensemble des stations distantes de moins de Xm
   --d'une station result donnee hormis la station result elle mÃªme
   --parametres:
   --databaseStation,in reseauStation, la base de donnees des stations
   --X,in float,la distance souhaitee
   --ref:in station,la station de reference
   --pre-condition:aucune
   --post-condition:les stations distantes de moins de Xm sont retournees
   --type de retour: reseauStation
   --exception:aucune
   --tests:
   --la distante X vaut 0, aucune station n'est retournee
   --la base de donnee r est vide, on retourne null
   -------------------------------------------------------------------------
   function stationDistanteX (databaseStation:in reseauStation;
      X:in float;ref:in station) return reseauStation is
      deb:reseauStation;--pointeur qui parcourt le reseau
      d:float;--distance
      debx,deby,x1,y1:float;--coordoonees
      tab:reseauStation;--accumulateur des stations a moins de Xm de ref
      s:station;
   begin
      x1:=ref.coordo.absc;
      y1:=ref.coordo.ordo;

      deb:=databaseStation;
      while deb/=null loop
         debx:=deb.all.val.coordo.absc;
```

```
        deby:=deb.all.val.coordo.ordo;
        d:=sqrt((x1-debx)**2+(y1-deby)**2);
        if (d>0.0 and d<=X)
        then
           s:=deb.all.val;
           inserer_en_tete(tab,s);
           deb:=deb.all.suivant;
        else
           deb:=deb.all.suivant;
        end if;
     end loop;
     --deb=null
     --tab contient les stations a moins de Xm de la station ref
     return tab;
  end stationDistanteX;
  --
  --
  ------------------------------------------------------------------------
  --semantique:procedure afficheStationLibre
  --affiche les stations du reseau r ayant une place de parking libre
  --parametre:
  --r,in reseauStation
  --precondition:aucune
  --postcondition:les numeros de stations sont affiches
  --exception:aucune
  --tests:
  -- r est vide
  --r ne contient aucune station libre
  --r a au moins une station libre
  ------------------------------------------------------------------------
  procedure afficheStationLibre(r:in reseauStation) is
  begin
     if (r=null)
     then
        put("]");
     elsif not(r.all.val.placeLibre)
     then
        afficheStationLibre(r.all.suivant);
     else
        put(r.all.val.num,3);
        put("->");
        afficheStationLibre(r.all.suivant);
     end if;
  end afficheStationLibre;
  --
  --
  ------------------------------------------------------------------------
  --semantique: enlever un element e de la liste l (liste vide ou non vide)
  --parametres: l donnee/resultat type listeVelo
  --e donnee type integer
  --pre-condition: aucune
  --post-condition: e n'appartient pas a la liste
  --exception: aucune
  ------------------------------------------------------------------------
  procedure enlever(l:in out reseauStation; e:in integer) is
     p:reseauStation;
     q:reseauStation;
  begin
     if (l/=null and then l.all.val.num=e)
     then
        l:=l.all.suivant;
     elsif (l=null)
```



```
     then
        null;
     else
        p:=l;
        q:=l;
        while (p/=null and then p.all.val.num/=e) loop
           q:=p;
           p:=p.all.suivant;
        end loop;
        --p=null ou p.all.val=e
        if (p=null)
        then
           null;
        else
           q.all.suivant:=p.all.suivant;
        end if;
     end if;
  end enlever;
  --
  --
  ------------------------------------------------------------------------
  --semantique:procedure statStation
  --donne un compte rendu concernant chacune des stations du reseau
  --coordonnees, velos,le numero etc
  --parametre:
  --r:in reseauStation,la base de donnee des stations
  --precondition:aucune
  --postconditon:le compte rendu de chaque station est affiche
  --exeption:aucune
  --tests:
  --la base de donnee est vide
  --la base de donnee a au moins une station
  ------------------------------------------------------------------------
  procedure statStation (databaseStation:in reseauStation) is
     bs:reseauStation;
  begin
     cursor_color(3);
     put_line("****************************************");
     put_line("   Compte rendu des bases de donnees    ");
     put_line("****************************************");
     new_line;
     new_line;
     cursor_color(2);
     put_line("********** LES STATIONS **********");
     cursor_color(4);
     new_line;
     bs:=databaseStation;
     while not(est_vide (bs)) loop
        put_line("************************");
        put("station ");
        put(bs.all.val.num);
        new_line;
        put_line("************************");
        new_line;
        put_line("----------------------------");
        put_line("coordoonees:");
        put("abscisse: ");
        put(bs.all.val.coordo.absc,2,0,0);
        new_line;
        put("ordonnee: ");
        put(bs.all.val.coordo.ordo,2,0,0);
        new_line;
```

```
         put_line("------------------------------");
         put("nombre de places du parking: ");
         put(bs.all.val.nbVelos);
         new_line;
         put_line("------------------------------");
         put_line("stations a proximite:");
         afficher(bs.all.val.stations_1_km);
         new_line;
         put_line("avec place libre:");
         afficheStationLibre(bs.all.val.stations_1_km);
         new_line;
         put_line("------------------------------");
         put_line("---les velos---");
         put_line("velos disponibles:");
         afficher(bs.all.val.velosDispo);
         new_line;
         put("total velos: ");
         put(longueur(bs.all.val.velosDispo));
         new_line;
         bs:=bs.all.suivant;
         put_line("******************************");
      end loop;
      cursor_color(7);
      new_line;
   end statStation;

end p_station;
```

```
package body p_utilisateur is

    ------------------------------------------------------------------------
    --semantique:fonction listevide creer une liste vide
    --parametres: aucun
    --type-retour: listeUtilisateur
    --pre-condition: aucune
    --post-condition: est_vide (l) vaut vrai
    --exception: aucune
    ------------------------------------------------------------------------
    function listevide return listeUtilisateur is
    begin
        return null;
    end listevide;
    --
    --
    ------------------------------------------------------------------------
    --semantique: fonction est_vide teste si une liste l est vide
    --parametres: l donnee type listeUtilisateur
    --type retour: booleen
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    ------------------------------------------------------------------------
    function est_vide (l:in listeUtilisateur) return boolean is
    begin
        return (l=null);
    end est_vide;
    --
    --
    ------------------------------------------------------------------------
    --semantique: procedure inserer_en_tete
    --insere en tete de la listeNumstation l l'elt nouveau
    --parametres: l donnee/resultat type listeUtilisateur
    --nouveau donnee type utilisateur
    --pre-condition: aucune
    --post-condition: nouveau appartient a la liste
    --exception: aucune
    ------------------------------------------------------------------------
    procedure inserer_en_tete(l:in out listeUtilisateur; nouveau:in utilisateur) is

        pins:listeUtilisateur; --pointeur d'insertion
    begin
        pins:= new noeud'(nouveau,l);
        l:=pins;
    end inserer_en_tete;
    --
    --


    ------------------------------------------------------------------------
    --semantique: fonction rechercher recherche si e appartient a la liste l
    --retourne son adresse
    --ou null si la liste est vide ou si e n'appartient pas a la liste
    --parametres: l donnee type listeUtilisateur
    --e donnee type entier
    --precision:in string,precision de recherche(num,codePostal)
    --type-retour: listeUtilisateur
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    ------------------------------------------------------------------------
    function rechercher(l:in listeUtilisateur; e:in integer;precision:in string)
```



```
        return listeUtilisateur is
        courant:listeUtilisateur;
    begin
        courant:=l;
        if precision="num"
        then
            while (courant/=null and then courant.all.val.num/=e) loop
                courant:=courant.all.suivant;
            end loop;
        elsif precision="codePostal"
        then
            while (courant/=null and then courant.all.val.codePostal/=e) loop
                courant:=courant.all.suivant;
            end loop;
        else
            null;
        end if;
        --courant.all.val.precision=e ou courant=null
        return courant;
    end rechercher;
    --
    --
    ------------------------------------------------------------------------
    --semantique: fonction statUser
    --donne un compte rendu de chacun des abonnes
    --parametres:
    --databaseUser,in listeUtilisateur, base de donnees utilisateurs
    --pre-condition: aucune
    --post-condition: aucune
    --exception: aucune
    --tests:
    --la base de donnee est vide
    --la base de donnee a au moins un utilisateur
    ------------------------------------------------------------------------
    procedure statUser (databaseUser:in listeUtilisateur) is
        bu:listeUtilisateur;
        cmpt:integer;
    begin
        new_line;
        cursor_color(2);
        cmpt:=1;
        put_line("********** LES ABONNES **********");
        new_line;
        new_line;
        cursor_color(4);
        bu:=databaseUser;
        while bu/=null loop
            put_line("************************");
            put("utilisateur:");
            put(cmpt);
            new_line;
            put_line("************************");
            put("numero d'identifiant: ");
            put(bu.all.val.num,4);
            new_line;
            put("boÃ®te postale: ");
            put(bu.all.val.codePostal,11);
            new_line;
            put("Credit: ");
            put(bu.all.val.credit,10,0,0);
            put(" euros");
            new_line;
```

```
            put("Caution: ");
            put(bu.all.val.caution);
            put(" euros");
            new_line;
            put("velos en votre possession : ");
            afficher(bu.all.val.veloPris);
            new_line;
            cmpt:=cmpt+1;
            bu:=bu.all.suivant;
            put_line("-----------------------------");
        end loop;
        cursor_color(7);
        new_line;
    end statUser;
    --
    ------------------------------------------------------------------------
  --fonction recup<<nomduchamp>>
  --recupere le champ <<nomduchamp>>
  --parametres
  --userlist,in listeUtilisateur
  --precondition aucune
  --postcondition aucune
  --type de retour:le type du champ recupere
  --exception:userlist est vide
  ------------------------------------------------------------------------
    function recupCredit(userlist:in listeUtilisateur) return float is
    begin
        if userlist=null
        then
            raise exception_compte;
        else
            return userlist.all.val.credit;
        end if;
    end recupCredit;

    function recupCaution(userlist:in listeUtilisateur) return integer is
    begin
        if userlist=null
        then
            raise exception_compte;
        else
            return userlist.all.val.caution;
        end if;
    end recupCaution;

  function recupVeloPris(userlist:in listeUtilisateur) return listeVelo is
    begin
        if userlist=null
        then
            raise exception_velo;
        else
            return userlist.all.val.veloPris;
        end if;
    end recupVeloPris;

    function recuptempsprise(userlist:in listeUtilisateur) return integer is
    begin
        if userlist= null
        then
            raise exception_compte;
        else
            return userlist.all.val.tempsprise;
```



```
        end if;
    end recuptempsprise;
  ------------------------------------------------------------------------
  --
  --
  ------------------------------------------------------------------------
   --procedure set<<nomduchamp>>
   --met a jour le champ <<nomduchamp>>
   --parametres
   --s,in listeUtilisateur
   --le(s) autres parametres (in) representent
   --la valeur avec laquelle on souhaite mettre a jour
   --precondition aucune
   --postcondition aucune
   --exception:userlist est vide
   ------------------------------------------------------------------------
    procedure setCredit(userlist:in listeUtilisateur;x:in float)is
    begin
        if userlist=null
        then
            raise exception_compte;
        else
            userlist.all.val.credit:=x;
        end if;
    end setCredit;

    procedure setCaution (userlist:in listeUtilisateur;x:in integer) is
    begin
        if userlist=null
        then
            raise exception_compte;
        else
            userlist.all.val.caution:=x;
        end if;
    end setCaution;

    procedure setVeloPris(userlist:in listeUtilisateur;velo_list:in listeVelo)is
    begin
        if userlist=null
        then
            raise exception_velo;
        else
            userlist.all.val.veloPris:=velo_list;
        end if;
    end setVeloPris;

    procedure settempsprise(userlist:in listeUtilisateur;temps:in integer)is
    begin
        if userlist=null
        then
            raise exception_compte;
        else
            userlist.all.val.tempsprise:=temps;
        end if;
    end settempsprise;
  ------------------------------------------------------------------------
---
  --
  --
  ------------------------------------------------------------------------
--
    --procedure setAll
```

```
   --met a jour tous les champs d'un utilisateur
   --parametres
   --user,in out utilisateur
   --le(s) autres parametres (in) representent
   --la valeur avec laquelle on souhaite mettre a jour
   --precondition aucune
   --postcondition aucune
   --exception:aucune
   procedure setAll(user:in out utilisateur;num:in integer;BP:in integer;
       credit:in float;caution:in integer;velos:in listeVelo;temps:in integer)is
    begin
        user:=(num,BP,credit,caution,velos,temps);
    end setAll;
  ------------------------------------------------------------------------------
---

end p_utilisateur;
```

```
package body p_menu is

    --Semantique: fonction qui affichage le menu de faÃ§on conviviale et fiable
    --puis recupere le choix entre au clavier.
    --parametres:aucun
    --exception:aucune
    --type de retour:caractere

    function choixMenu return character is
        choix:character;
    begin
        loop
            cursor_color(4);

            put_line("-------------------------Administrateur--------------------");
            cursor_color(2);
            new_line;
            put_line("(+). Creer une station");
            put_line("(-). Detruire une station");
            put_line("(*). Consulter base de donnees");
            put_line("-----------------------------------------------------------");
            new_line;
            cursor_color(4);
            put_line("***********************************************************");
            put_line("Chers utilisateurs, Bienvenue dans le reseau VeloTlse Â®");
            put_line("Choisissez une option.");
            put_line("***********************************************************");
            new_line;
            put_line("-----------------------Utilisateurs-----------------------");
            new_line;
            cursor_color(2);
            put_line("a. S'abonner a VeloToulouse");
            put_line("b. Prendre un velo");
            put_line("c. Restituer un velo");
            put_line("d. Acceder a son compte");
            put_line("-----------------------------------------------------------");
            new_line;
            cursor_color(4);
            put_line("----------------------Stations----------------------------");
            new_line;
            cursor_color(2);
            put_line("e. Consulter la mini carte");
            put_line("f. Stations distantes de moins de 1km avec parking libre");
            put_line("g. Stations a X m(a preciser) d'une station de reference");
            put_line("q. Quitter");
            put_line("-----------------------------------------------------------");
            new_line;

            cursor_color(7);
            new_line;
            new_line;
            new_line;

            get (choix);
            exit when (choix ='q'or choix='a' or choix='b' or choix='c' or choix
='d'
            or choix='e' or choix='f' or choix='g' or choix='+' or choix='-' or
choix='*');
        end loop;
        --choix appartient a la liste des caracteres souhaites(a,b,c,etc...)
        return choix;
    end choixMenu;
```



```
    --
    --
    ------------------------------------------------------------------------
    --semantique: procedure creeStation
    --cree une nouvelle station
    --parametres:
    --databaseVelo,in out listeVelo,base de donnees des velos deja crees
    --les numeros des velos sont generes automatiquement
    --chaque velo a un numero unique
    --on a besoin de cette base de donnees pour eviter les repetitions
    --preconditions:
    --le numero de la station est unique
    --postcondition:
    --le reseau a une station de plus
    --exception:aucune
    --  bool:=longueur(s.velosDispo)=nbPlace;
    ------------------------------------------------------------------------
    procedure creeStation(databaseStation:in out reseauStation;
        databaseVelo:in out listeVelo) is
        s:station;
        temp:reseauStation;
        pass:character;
        absc,ordo:float;
        nbPlace:integer;
        nbreVelo:integer;
        velosGeneres:listeVelo;
    begin

        put("mot de passe administrateur: ");
        get(pass);
        if (pass/='\')
        then
            put("cette application est reservee a l'administration");
        else
            loop
                put("Nombres de places du parking(>0): ");
                get(nbPlace);
                exit when (nbPlace>0);
            end loop;
            loop
                put("Nombres de Velos:(entre 0 et ");
                put(nbPlace,1);
                put("): ");
                get(nbrevelo);
                exit when (nbrevelo>=0 and nbrevelo<=nbPlace);
            end loop;

            put_line("coordonnees de la station en metres: ");
            put("abscisse: ");
            get(absc);
            put("ordonnee:");
            get(ordo);

            temp:=rechercher(databaseStation,absc,ordo);
            if (est_vide(databaseStation) or (not(est_vide(databaseStation)) and
then est_vide(temp)))
            then
                if est_vide(databaseStation)
                then
                    setNum(s,1);
                else
                    setNum(s,recupNum(databaseStation)+1);
```

```
                end if;
                setNbVelos(s,nbPlace);
                setCoordo(s,absc,ordo);
                genereVelos((nbrevelo),databaseVelo,velosGeneres);
                setVelosDispo(s,velosGeneres);
                setPlaceLibre(s,(longueur(recupVelosDispo(s))/=recupNbVelos(s)))
;
                inserer_en_tete(databaseStation,s);
                setStations(databaseStation,1000.0);
                put_line("operation effectuee.");
                put("numero de station creee: ");
                put(recupNum(s));
            else
                put_line("echec de l'operation");
                put_line("une station existe deja a cet emplacement");
            end if;
        end if;
        new_line;
        new_line;
    end creeStation;
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure enleveStation
    --enleve une station du reseau sans perube son fonctionnement
    --parametres:databaseStation,in reseauStation,base de donnee
    --pre:reseau non vide
    --post:le reseau a une station en moins
    --exception:aucune
    ------------------------------------------------------------------------
    procedure enleveStation(databaseStation:in out reseauStation) is
        result:reseauStation;--adresse de la station a detruire
        choixStation:integer;--numero de la station a detuire
        pass:character;--mot de passe administrateur
    begin
        put("mot de passe administrateur: ");
        get(pass);
        if (pass/='\')
        then
            put_line("Echec de l'operation");
            put("cette application est reservee a l'administration");
        else
            if est_vide(databaseStation)
            then
                put("Oups!! Aucune station n'a ete creee");
            else
                put_line("Quelle station voulez vous detruire?");
                afficher(databaseStation);
                new_line;
                loop
                    put("saisir un numero figurant dans la liste proposee:  ");
                    get (choixStation);
                    result:=rechercher(databaseStation,choixStation);
                    exit when (not est_vide(result));
                end loop;
                --result est l'adresse d'une station
                --choixStation est le numero de la station a detruire
                enlever(databaseStation,choixStation);
                setStations(databaseStation,1000.0);
                put_line("operation effectuee.");
                put("numero de station enlevee: ");
                put(choixStation);
```



```
            end if;
        end if;
        new_line;
        new_line;
    end enleveStation;
    --
    --
    ------------------------------------------------------------------------
    --procedure consulteBaseDedonnees
    --donne un compte rendu des bases de donnees
    --parametres:
    --r,in reseauStation,base de donnee des stations
    --lu:in listeUtilisateur,base de donnee des utilisateurs
    ------------------------------------------------------------------------
    procedure consulteBaseDedonnees(r:in reseauStation;lu:in listeUtilisateur)is
        pass:character;--mot de passe administrateur
    begin
        put("mot de passe administrateur: ");
        get(pass);
        if (pass/='\')
        then
            put("cette application est reservee a l'administration");
            new_line;
            new_line;
        else
            statStation(r);
            statUser(lu);
            new_line;
        end if;
    end consulteBaseDedonnees;
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure abonnement
    --ajoute un utilisateur a la base de donnees utilisateurs.
    --parametres:
    --databaseUser:la base de donnees utilisateurs
    --preconditions:
    --avoir une caution de 150 euros et un credit de 3 euros
    --postcondition: un utilisateur est ajoute a la base de donnees.
    --exception: aucune
    ------------------------------------------------------------------------
    procedure abonnement(databaseUser:in out listeUtilisateur) is
        num:integer; --numero d'utilisateur
        BP:integer;--code postal
        answer:character;--pour voir s'il es en regle pour s'abonner
        user:utilisateur;--utilisateur qui s'abonne
        research:listeUtilisateur;--resultat de recherche ds la base de donnee
    begin
        put("numero utilisateur: ");
        get (num);
        put("Boite Postale:");
        get (BP);

        --eviter qu'il y'ait deux abonnes avec le meme numero d'identifiant
        research:=rechercher(databaseUser,num,"num");
        if not(est_vide(research))
        then
            put_line("numero invalide.");
            put_line("Pour vous abonnez, munissez vous d'un autre numero aupres de notre agence");
        else
            --verifier qu'il rempli les conditions pour un abonnement
```

```
            loop
                put("Disposez vous d'une caution de ");
                put(cautionInitiale,1);
                put(" euros et d'un credit d'au moins ");
                put(creditInitial,2,0,0);
                put_line(" euros? oui(O)/non(N) ");
                get (answer);
                exit when (answer='o' or answer='O' or answer ='n' or answer ='
N');
            end loop;
            -- answer appartient a la liste souhaitee
            if (answer='n' or answer='N')
            then
                put_line("Desole, vous ne remplissez pas les conditions pour l'abonnement.");
            else
                setAll(user,num,BP,creditInitial,cautionInitiale,listevide,0);
                inserer_en_tete(databaseUser,user);
                put_line("Operation effectuee Merci");
                put(num,3);
                put_line(" Vous avez ete abonne. Vous pouvez utiliser le veloToulouse! ");
            end if;
        end if;
        new_line;
        new_line;
    end abonnement;
    --
    --
    ------------------------------------------------------------------------------
    --semantique:procedure prendreVelo
    --autorise un abonne a prendre un velo
    --parametres:
    --databaseVelo,in out listeVelo, base de donnees des velos
    --databaseUser,in out listeUtilisateur, base de donnees des utilisateurs
    --databaseStation:in reseauStation,base de donnees des stations
    --preconditions:
    --avoir un numero valide (on suppose qu'un numero est valide s'il
    --figure pas dans la base de donnees)
    --avoir un credit d'au moins 3 euros
    --velo disponible dans la station
    --postcondition: un capteur signale l'absence du velo pris
    --dans la base de donnees de velos et
    --le numero de velo s'ajoute a la liste de velos pris par l'utilisateur
    --exception: aucune
    ------------------------------------------------------------------------------
    procedure prendreVelo(databaseVelo:in out listeVelo;
        databaseUser:in listeUtilisateur;databaseStation:in
        reseauStation) is
        num:integer;
        BP:integer;
        researchnum:listeUtilisateur;--resultat de la recherche du num
        researchBP:listeUtilisateur;--resultat de la recherche de la BP
        choixVelo,caution:integer;--numero de velo choisi
        choixStation:integer;--numero de la station choisie
        result,temp:reseauStation;--resultat d'une recherche de station
        result2:listeVelo;--resultat d'une recherche de velo
        velolist,numsvelo:listeVelo;
        list_velo:listeVelo;
        solde:float;--credit de l'utilisateur;
        ans:character;--reponse
        a,e:integer;--pour afficher la date et l'heure de la prise du velo

    begin
```



```
        put("numero utilisateur: ");
        get (num);
        put("Boite Postale:");
        get (BP);
        --verifier que les identifiants sont corrects
        researchnum:=rechercher(databaseUser,num,"num");
        researchBP:=rechercher(researchnum,BP,"codePostal");
        --verifier que son solde et sa caution lui permettent de prendre un velo
        --researchBP
        solde:=recupCredit(researchBP);
        caution:=recupCaution(researchBP);
        if (solde<=0.0 or caution =0)
        then
            put_line("Desole.Vous ne pouvez prendre de velo si votre compte n'est pas credite");
        else
            numsvelo:=recupVeloPris(researchBP);
            if not (est_vide(numsvelo))
            then
                put_line("vous etes encore en possession d'un velo");
            elsif est_vide(databaseStation)
            then
                put("Oups!! Aucune station n'a ete creee");
            else
                --on lui demande la station oÃ¹ il veut prendre le velo
                --et le velo qu'il desire

                put_line("Dans quelle station voulez vous prendre un velo?");
                afficher(databaseStation);
                new_line;
                loop
                    put("saisir un numero figurant dans la liste proposee: ");
                    get (choixStation);
                    result:=rechercher(databaseStation,choixStation);
                    exit when (not est_vide(result));
                end loop;
                --result est l'adresse de la station choisie
                new_line;
                put_line("quel velo voulez vous prendre?");
                velolist:=recupere(result);
                afficher(velolist);
                new_line;
                if est_vide(velolist)
                then
                    put_line("Oups!!la station a ete cree sans parking a velo");
                    put_line("Ou alors ne dispose plus de velos");
                else
                    loop
                        put("saisir un numero figurant dans la liste proposee: ");
                        get(choixVelo);
                        result2:=rechercher(velolist,choixVelo);
                        exit when (not est_vide(result2));
                    end loop;
                    --result2 est l'adresse du velo choisi

                    --on enleve le velo choisi de la station
                    --et on lui donne ce velo
                    list_velo:=recupVeloPris(researchBP);
                    inserer_en_tete(list_velo,choixVelo);
                    setVeloPris(researchBP,list_velo);
                    enlever(velolist,choixVelo);
                    set(result,velolist);
                    set(result,true);
```

```
                    setStations(databaseStation,1000.0);
                    put_line("Le velo est a votre disposition");
                    cursor_color(2);
                    put_line("A SAVOIR:");
                    put_line("Le credit est debite de 50 centimes par demi heure d'utilisation");
                    put_line("la premiere demi-heure est gratuite.");
                    put_line("velo pris le ");

                    put(integer(day(clock)),2);
                    put("/");
                    put(integer(month(clock)),2);
                    put("/");
                    put(integer(year(clock)),2);
                    new_line;
                    put("a ");
                    settempsprise(researchBP,integer(seconds(clock)));
                    a:=recuptempsprise(researchBP);
                    e:=a mod 3600;
                    put(a/3600,2);
                    put(" h ");
                    put(e/60,2);
                    put(" min ");
                    put(e mod 60,2);
                    put(" s");
                    cursor_color(7);

                    new_line;
                    new_line;
                    new_line;
                    --une fois en possession du velo,
                    --il pourrait eventuellement le voler
                    loop
                        put_line("(o_O)Â¹ une idee malsaine me traverse l'esprit ");
                        put_line("($_$)(^_^). Je vole le velo? (o/n)");
                        get(ans);
                        exit when (ans='o' or ans='n');
                    end loop;
                    --ans='o' ou ans='n'
                    if ans='o' --il a vole le velo et on coupe sa caution
                    then
                        setCaution(researchBP,0);
                        list_velo:=recupVeloPris(researchBP);
                        enlever(list_velo,choixVelo);
                        setVeloPris(researchBP,list_velo);
                    else --il n'a pas vole le velo
                        null;
                    end if;
                end if;
            end if;
        end if;
        new_line;
        new_line;

    exception
        when exception_compte=>put("identifiants incorrects");
        new_line;
        new_line;
        when others => put("autres exceptions");
        new_line;
        new_line;
    end prendreVelo;
    --
```



```
    ------------------------------------------------------------------------
    --semantique procedure restituerVelo
    --restitue un velo a une station
    --parametres:
    --databaseVelo,in out listeVelo, base de donnees des velos
    --databaseUser,in listeUtilisateur, base de donnees des utilisateurs
    --databaseStation:in reseauStation,base de donnees des stations
    --preconditions:
    --la station dispose d'au moins une place libre
    --postcondition:
    --un capteur signale qu'un velo a ete restituer
    --dans la base de donnees des velos
    --le credit est debite en fonction du temps passe avec le velo
    --exception:aucune.
    ------------------------------------------------------------------------
    procedure restituerVelo(databaseVelo:in out listeVelo;
        databaseUser:in listeUtilisateur; databaseStation:in
        reseauStation) is
        num:integer;
        BP:integer;
        researchnum:listeUtilisateur;--resultat de la recherche du num
        researchBP:listeUtilisateur;--resultat de la recherche de la BP
        numsvelo:listeVelo;
        velolist:listeVelo;
        choixVelo:integer;--numero de velo choisi
        result2:listeVelo;--resultat d'une recherche
        choixStation:integer;--numero de la station choisie
        result:reseauStation;--resultat d'une recherche de station
        bool,bool2:boolean;
        pass:character;--mot de passe de l'administration
        temps:integer;--nombre de minutes passe par l'abonne avec le velo
        sec:integer;--secondes
        solde:float;--solde qui sera debite
        ans:character;--reponse
    begin
        put("numero utilisateur: ");
        get (num);
        put("Boite Postale:");
        get (BP);
        researchnum:=rechercher(databaseUser,num,"num");
        researchBP:=rechercher(researchnum,BP,"codePostal");
        if est_vide(researchBP)
        then
            put_line("Identifiants incorrects.");
        else
            put_line("Dans quelle station voulez vous rendre le velo?");
            afficher(databaseStation);
            new_line;
            if est_vide(databaseStation)
            then
                put("Oups!! Aucune station n'a ete creee");
            else
                loop
                    put("saisir un numero figurant dans la liste proposee: ");
                    get (choixStation);
                    result:=rechercher(databaseStation,choixStation);
                    exit when (not est_vide(result));
                end loop;
                --result est l'adresse d'une station
                velolist:=recupere(result);
                bool:=longueur(velolist)=recupere(result);
                bool2:=bool;
```

```
                if bool
                then
                    put_line("Pas de place disponible dans cette station");
                else
                    numsvelo:=recupVeloPris(researchBP);
                    if est_vide(numsvelo)
                    then
                        put("Vous n'avez aucun velo a rendre!");
                    else
                        choixVelo:=recupVelo(numsvelo);
                        enlever(numsvelo,choixVelo);
                        setVeloPris(researchBP,numsvelo);
                        inserer_en_tete(velolist,choixVelo);
                        set(result,velolist);
                        set(result,bool2);
                        put_line("Merci de l'avoir rendu!");
                        setStations(databaseStation,1000.0);
                        new_line;
                        new_line;

                        --calcul du temps mis
                        temps:=(integer(seconds(clock))-recuptempsprise(research
BP))/60;
                        sec:=(integer(seconds(clock))-recuptempsprise(researchBP
)) mod 60;
                        solde:= -0.5*float'floor((-1.0/30.0)*float(temps))-0.5;
                        --float'floor(x) est la partie entiere de x
                        cursor_color(2);
                        put("nombre de temps passe : ");
                        put(temps,1);
                        put("min ");
                        put(sec,1);
                        put_line("s");
                        cursor_color(7);
                        put_line("reserve a l'administration:");
                        loop
                            put("mot de passe administrateur: ");
                            get(pass);
                            exit when (pass='\');
                        end loop;
                        --pass est le mot de passe
                        --correct
                        --pour les tests afin de gagner du temps j'utiliserai
                        --cette boucle, pour decider du temps qu'a mis l'abonne
                        --avec le velo
                        --loop
                            --put("temps(en min) effectue avec le velo: ");
                            --get(temps);
                            --exit when temps>0.0;
                        --end loop;
                        --temps>0

                        setCredit(researchBP,(recupCredit(researchBP)-solde));
                        --on verifie si le velo a ete deteriore
                        loop
                            put("velo deteriore (O/N)?: ");
                            get(ans);
                            exit when (ans='o' or ans='O'or ans ='n' or ans='N'
);
                        end loop;
                        --ans vaut {o,O,n ou N}
                        if (ans='o' or ans='O')
```



```
                        then
                            setCaution(researchBP,0);
                        else
                            null;
                        end if;
                    end if;
                end if;
            end if;
        end if;

        new_line;
        new_line;
    end restituerVelo;
    --
    --
    ------------------------------------------------------------------------------
    --semantique:procedure monCompte
    --autorise un abonne a consulter les details de son compte
    --parametres:
    --databaseUser,in listeUtilisateur, base de donnees des utilisateurs
    --preconditions:
    --avoir un numero valide (on suppose qu'un numero est valide s'il
    --figure pas dans la base de donnees)
    --postcondition: les infos sont affiches
    --exception: aucune
    ------------------------------------------------------------------------------
    procedure monCompte(databaseUser:in listeUtilisateur) is
        num:integer;
        BP:integer;
        researchnum:listeUtilisateur;--resultat de la recherche du num
        researchBP:listeUtilisateur;--resultat de la recherche de la BP
        solde:float;--solde qui sera debite
        ans:character;--reponse
    begin
        put("numero utilisateur: ");
        get (num);
        put("Boite Postale:");
        get (BP);
        researchnum:=rechercher(databaseUser,num,"num");
        researchBP:=rechercher(researchnum,BP,"codePostal");
        if est_vide(researchBP)
        then
            put_line("Identifiants incorrects.");
        else
            put("Patientez."); delay 1.0;
            put(".");delay 1.0; --Âßa c'est juste pour que le progamme
            put_line(".");delay 1.0;  --paraisse dynamique
            cursor_color(4);
            put_line("*****************************************");
            put_line("      Synthese de votre Compte      ");
            put_line("*****************************************");
            new_line;
            put("numero d'identifiant: ");
            put(num,4);
            new_line;
            put("boÃ®te postale: ");
            put(BP,11);
            new_line;
            put("Credit: ");
            put(recupCredit(researchBP),10,0,0);
            put(" euros");
            new_line;
```

```
            put("Caution: ");
            put(recupCaution(researchBP));
            put(" euros");
            new_line;
            put("velos en votre possession : ");
            afficher(recupVeloPris(researchBP));
            new_line;
            cursor_color(7);
            loop
                put_line("Voulez vous creditez votre compte?oui(o)/non(n)");
                get(ans);
                exit when (ans='o' or ans='O'or ans ='n' or ans='N');
            end loop;
            --ans vaut {o,O,n ou N}
            if (ans='o' or ans='O')
            then
                put("saisir le montant a ajouter: ");
                get(solde);
                setCredit(researchBP,(solde+recupCredit(researchBP)));
                setCaution(researchBP,150);
            else
                null;
            end if;
            cursor_color(4);
            put_line("*****************************************");
            cursor_color(7);
        end if;
        new_line;
        new_line;
    end monCompte;
    --
    --
    ------------------------------------------------------------------
    --semantique:procedure miniCarte affiche les stations
    --du reseau a leur emplacement (en fonction de leurs coordoonees
    --parametres:databaseStation:in reseauStation,la base de donnee
    --precondition:aucune
    --postcondition:stations affichees
    --exception:aucune
    ------------------------------------------------------------------
    procedure miniCarte (databaseStation:in reseauStation) is
    begin
        put_line("L'ensemble des stations du reseau:");
        clear_screen;
        affichage(databaseStation);
        clear_screen;
    end miniCarte;
    --
    --
    ------------------------------------------------------------------------
    --semantique:procedure stationDistanteXm
    --affiche l'ensemble des stations distantes de moins de Xm
    --parametres: databaseStation:in reseauStation,la base de donnees
    --precondition:la base de donnee n'est pas vide
    --postcondition:les stations sont affichees
    --exception:aucune
    ------------------------------------------------------------------------
    procedure stationDistanteXm (databaseStation:in reseauStation) is
        result:reseauStation;--adresse de la station de reference
        choixStation:integer;--le choix de la station de reference
        tab:reseauStation;--accumulateur qui contiendra les stations a afficher
        X:float;--la distance
```



```
    begin
        if est_vide(databaseStation)
        then
            put("Oups!! Aucune station n'a ete creee");
        else
            put_line("Quelle est la station de reference?");
            afficher(databaseStation);
            new_line;
            loop
                put("saisir un numero figurant dans la liste proposee: ");
                get (choixStation);
                result:=rechercher(databaseStation,choixStation);
                exit when (not est_vide(result));
            end loop;
            --result est l'adresse d'une station
            --choixStation est le numero de la station de reference
            put("nombre de metres: X= ");
            get(X);
            tab:=stationDistanteX (databaseStation, X, recupVal(result));
            put("Les stations a moins de ");
            put(X,2,0,0);
            put(" metres de la station");
            put(choixStation,2);
            put_line(" sont:");
            afficher(tab);
        end if;
        new_line;
        new_line;
    end stationDistanteXm;
    --
    --
    ------------------------------------------------------------------
    --semantique:procedure parkingsLibre
    --affiche les stations ayant une place de parking libre
    --parametres:databaseStation,in reseauStation,base de donnee
    --pre:base de donnee non vide
    --post:stations affichees
    --exception:aucune
    ------------------------------------------------------------------
    procedure parkingsLibre(databaseStation:in reseauStation) is
        result:reseauStation;--adresse de la station de reference
        choixStation:integer;-- numero de la station de reference
    begin
        if est_vide(databaseStation)
        then
            put("Oups!! Aucune station n'a ete creee");
        else
            put_line("Dans quelle station vous trouvez vous?");
            afficher(databaseStation);
            new_line;
            loop
                put("saisir un numero figurant dans la liste proposee: ");
                get (choixStation);
                result:=rechercher(databaseStation,choixStation);
                exit when (not est_vide(result));
            end loop;
            --result est l'adresse d'une station
            --choixStation est le numero de la station de reference
            new_line;
            put_line("Les stations a moins de 1km disposants d'un parking libre:");
            afficheStationLibre(recupStations_1_km(recupVal(result)));
        end if;
```

```
        new_line;
        new_line;
    end parkingsLibre;
    --
    --

    ------------------------------------------------------------------------
    --procedure quitter
    --quitte le menu en laissant un message et en arretant la boucle
    --qui reaffiche le menu
    --parametre:bool,in boolean, la condition d'arret de la boucle
    --pre,post,exception:aucun
    ------------------------------------------------------------------------
    procedure quitter (bool:in out boolean) is
    begin
        put("Merci de votre visite. Au revoir");
        bool:=false;
        new_line;
        new_line;
    end quitter;
    ------------------------------------------------------------------------
end p_menu;
```

```
with libansi;use libansi;
with p_menu;use p_menu;
with p_station; use p_station;
with p_velo; use p_velo;
with p_utilisateur; use p_utilisateur;
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;
with ada.float_text_io; use ada.float_text_io;
with Ada.Numerics.Elementary_Functions; use  Ada.Numerics.Elementary_Functions;


procedure velotlse is

    repeat:boolean;--on reaffichera le menu tant que le choix n'est pas 'quit'.
    listUser:listeUtilisateur;--base de donnees des utilisateurs
    listVelo:listeVelo;--base de donnees des velos
    velosGenere:listeVelo;
    listStation:reseauStation;--base de donnee des stations

begin
    clear_screen;
    listUser:=listevide;
    listStation:=listevide;
    listVelo:=listevide;
    velosGenere:=listevide;
    repeat:=true;
    while repeat  loop
        begin
            case choixMenu is
                when '+'=>  creeStation(listStation,listVelo);

                when '-'=>  enleveStation(listStation);

                when '*'=>  consulteBaseDeDonnees(listStation,listUser);


                when 'a'=>  abonnement(listUser);

                when 'b'=>  prendreVelo(listVelo,listUser,listStation);

                when 'c'=>  restituerVelo(listVelo,listUser,listStation);

                when 'd'=>  monCompte(listUser);

                when 'e'=>  miniCarte(listStation);

                when 'f'=>  parkingsLibre(listStation);

                when 'g'=>  stationDistanteXm (listStation);

                when 'q'=>  quitter(repeat);

                when others=> null; -- ce cas deja filtre ne risque pas d'arrive
r
            end case;
            --repeat = false avec l'appel de quitter
        exception
            when exception_reseau=>put("acces a un reseau vide");
            when exception_velo=>put("acces a une liste de velo vide");
            when exception_compte=>put("acces a une liste d'utilisateur vide");
            when data_error=>put_line("entrer un chiffre SVP");
            when others=>put_line("une erreur s'est produite");
```



```
        end;

    end loop;
end velotlse;
```