Ecole **I**nternationale des **S**ciences du **T**raitement de l'**I**nformation
Classe : 1ière année

Examen de programmation ADA Groupe 2

Tous les documents sont autorisés

Exercice 1 : type, sous type, tableau, record, liste

Une entreprise de transport (bus) possède une liste de lignes de transport. Une ligne de bus est décrite par le numéro de la ligne, la liste des stations composant cette ligne et un tableau de contrats par semaine des différents chauffeurs (tableau ci-dessous). Une ligne de bus ne peut avoir plus de 30 chauffeurs.
Une station d'une ligne de bus est connue par son nom (maximum de 20 caractères). Un chauffeur est décrit complètement par son nom, son prénom (maximum de 20 caractères), son identificateur (un entier positif). Le contrat de travail par semaine d'un chauffeur donné est défini par les informations sur le chauffeur et par un tableau d'heures effectué par jour sur la semaine (voir exemple ci-dessous).
La charge de travail est calculée au quart d'heure près (5h=5, 5h15'= 5.25, 5h30=5.50, 5H45=5.75…).

	Lundi	**Mardi**	**Mercredi**	**Jeudi**	**Vendredi**
125 Charlie Cedric	6.0	5.75	5.0	4.5	4.25

Exemple d'un contrat de travail par semaine d'un chauffeur donné

1) Définir proprement toutes les types des structures de données qui interviennent dans le problème (Pas de valeur numérique dans les types, que des constantes). Heure, Jour, Chauffeur, Contrat, et Ligne_Bus,

-- *Nombre maximal de caracteres pour le nom, le prenom d'un chaffeur et le nom d'une station*
Max_longueur_Nom: **constant** Integer := 20; **(0.5pt)**

-- *Nombre maximal d'eleves*
Max_Nombre_Chauffeurs: **constant** Integer := 30; **(0.5pt)**

-- *Type pour représenter les heures effectuees par jour*
type T_Nb_Heures_Effectuees **is delta** 0.25 **range** 0.0..6.0; **(0.5pt)**

-- *Type jours ouvrables*
type T_Jours_Travail **is** (Lundi, Mardi, Mercredi, Jeudi, Vendredi); **(0.5pt)**

-- *Type nombre d'heures effectuees par semaine*

type T_Table_Heures_Semaine **is array**(T_Jours_Travail) **of** T_Nb_Heures_Effectuees; **(0.5pt)**

-- *Type pour représenter un employe* **(0.5pt)**

type T_Chauffeur **is record**

Identificateur : Positive;

Nom : String(1..Max_longueur_Nom);

Prenom : String(1..Max_longueur_Nom);

end record;

-- *type station*

type T_Station **is** String(1.. Max_longueur_Nom) ; **(0.5pt)**

-- *Type d'une liste de stations T_Listes_Stations* **(0.5pt)**

type Element ;
type Liste **is access** Element ;
type Element **is record**
Station : T_Station;
Suivant : Liste ;
end record ;

-- *Type d'un contrat d'un employe-heures par semaine* **(0.5pt)**

type T_Contrat is **record**

Chauffeur : T_Chauffeur;

Tab_Heures_Semaine : T_Table_Heures_Semaine;

end record;

-- Tableau employe-heures **(0.5pt)**

Type T_Tableau_Contrats **is array**(1..Max_Nombre_Chauffeurs) **of** T_Contrat;

type T_Ligne **is record** **(0.5pt)**

Numero : Positive;

Liste_Stations : Liste;

Tableau_Contrats : T_Tableau_Contrats ;

end record;

2. Ecrire une fonction qui vérifie si une station donnée fait partie ou pas d'une ligne de bus **(1pt)**

```
function Station_In_Ligne (Station: T_Station; Ligne: T_Ligne) return Boolean is
    Reponse : Boolean := False;
    Aux : Liste := Ligne.Liste_Stations;
begin  -- Station_In_Ligne
   -- Parcourir toute la classe
      while Aux /= null loop
            if (Station = Aux.Station) then
                  Reponse=True;
                  exit,
            else
                  Aux := Aux.all.Suivant;
            end if;
      end loop;
      return Reponse;
end Station_In_Ligne;
```

3) En utilisant la fonction définie dans 2, écrire une fonction permettant de savoir si une matière **ne fait pas partie** en même temps dans deux filières différentes. **(0.5pt)**

```
function Station_In_2_Lignes (Station: T_Station; Ligne1, Ligne2: T_Ligne) return Boolean is
begin  -- Station_In_2_Lignes
   return not (Station_In_Ligne (Station, Ligne1) and Station_In_Ligne (Station, Ligne2));
end Station_In_2_Lignes;
```

4) Pour une ligne donnée ***Ligne1,*** on veut éclater la liste de ses stations en deux listes. Une liste de stations qui appartiennent uniquement à cette ligne, et une autre liste de stations communes à une autre ligne de transport ***Ligne2 (***Utiliser la fonction de la question 3).

a) réfléchir à différentes solutions, et les expliquer ***(1.5pts)***

1. recopier les stations n'appartenant pas à une autre ligne pour les insérer dans une liste et même chose pour les stations communes à une autre ligne. La liste des stations de la ligne initiale n'est pas modifiée. L'algorithme doit nécessairement comporter des new … puisqu'il y aura création de nouveaux éléments.
2. modifier le chaînage de la liste initiale. La liste initiale sera perdue ; les éléments ne sont pas recrées mais chaînés différemment.
3. faire une liste de liste.

b)modifier selon le schéma ci-dessous le chainage de la liste initiale. La liste des stations de la ligne initiale ***Ligne1*** sera perdue, les éléments ne sont pas recrées mais chainés différemment. L'insertion doit se faire en tête de liste

Insertion en tête : pas de cas particulier (liste vide ou pas); pas besoin de parcourir la liste pour trouver la fin. Quand on a le choix, c'est en général l'insertion que l'on choisit. Ce qui donne avec les déclarations de la question 1 : ***(2pts)***

```
procedure Eclater (Ligne1 : in out Liste;
                   Ligne2 : in Liste ;
                   ListeStationsAppartenent : out Liste ;
                   ListeStationsNappartenentPas : out Liste) is
    Le_suivant : liste ;
begin
    ListeStationsAppartenent := null;
    ListeStationsNappartenentPas := null;
    Aux :=Ligne1;
    while Aux /=null loop – tant que je n'ai pas termine le parcours de la ligne Ligne1
        Le_Suivant :=Aux.all.suiv ; -- car on va perdre le chainage
        if  (Station_In_2_Lignes (Aux.all.Station,  Aux, Ligne2 ) ) then
            -- ajout en debut de la liste des station n'appartenant pas à une autre ligne
            aux.all.suivant:= ListeStationsNappartenentPas;
            ListeStationsNappartenentPas := Aux;
        else
            -- ajout en debut de la liste des impairs
            aux.all.suivant:= ListeStationsAppartenent;
            ListeStationsAppartenent := Aux;
        end if;
        Aux := Le_Suivant; -- pour passer a l element suivant
    end loop;
end Eclater ;
```

Exercice 2 : package, fichier, exception, tableau

Soit la spécification du package suivante :

1. Compléter cette spécification ***(2pts)***
2. Ecrire le corps de ce paquetage.

```
package Matrice is

  -- definition d'un sous type
  subtype T_Note is Float range 0.0..20.0;

 --  exception si le fichier est vide
 Fichier_Vide_Exception : exception;

  -- Une variable fichier texte
  Fichier_Traite  : File_Type ;
```

```
   -- definition de la matrice - non contrainte -
   type T_Matrice is array (Natural range <>, Natural range <>) of T_Note;

  -- Ouvrir un fichier texte en mode lecture et ecriture (2pts)
  procedure Open(File …..

  -- Lire les donnees à partir d'un fichier(2pts)
  function Lire (...;

   -- Transposee du tableau de note(2pts)
   function Transposer_Matrice (…;

  -- Ecrire les donnees dans le meme fichier sans ecraser les données initiales(2pts)
  procedure Ecrire (....;

end Matrice;
```