**Université du Québec à Montréal**

**TP 2**

**Gestion de cours et d’étudiant.**

**INF 2110-10**

**Été 2003**

Yann Bourdeau

BOUY06097202

Table des matières

[Conception Détaillé. 3](#_Toc46204443)

[1.1 Module Liste (liste\_dyn.ads) 3](#_Toc46204444)

[Creer\_Liste 3](#_Toc46204445)

[Liste\_Vide 3](#_Toc46204446)

[Detruire\_liste 4](#_Toc46204447)

[Nombre\_Elements 4](#_Toc46204448)

[Inserer\_en\_Ordre 5](#_Toc46204449)

[Modifier 6](#_Toc46204450)

[Obtenir 6](#_Toc46204451)

[Retirer 7](#_Toc46204452)

[Dernier 8](#_Toc46204453)

[Successeur 8](#_Toc46204454)

[Predecesseur 9](#_Toc46204455)

[Sauvegarder 9](#_Toc46204456)

[Recuperer 10](#_Toc46204457)

[Annexe A – Listage des sources. 11](#_Toc46204458)

[TP2.ADA 11](#_Toc46204459)

[LECTURE.ADS 26](#_Toc46204460)

[LECTURE.ADB 27](#_Toc46204461)

[LISTE\_DYN.ADS 29](#_Toc46204462)

[LISTE\_DYN.ADB 31](#_Toc46204463)

[LISTES\_TAB.ADS 36](#_Toc46204464)

[LISTES\_TAB.ADB 38](#_Toc46204465)

[testn1.txt 44](#_Toc46204466)

[testn2.txt 44](#_Toc46204467)

[testn3.txt 44](#_Toc46204468)

[testn4.txt 45](#_Toc46204469)

[testn5.txt 45](#_Toc46204470)

[testn6.txt 46](#_Toc46204471)

[testn7.txt 46](#_Toc46204472)

[testn81.txt 46](#_Toc46204473)

[testn82.txt 47](#_Toc46204474)

[testl1.txt 47](#_Toc46204475)

[testl2.txt 47](#_Toc46204476)

[testl3.txt 47](#_Toc46204477)

[testl4.txt 48](#_Toc46204478)

[testh1.txt 48](#_Toc46204479)

[testh2.txt 49](#_Toc46204480)

[testh3.txt 49](#_Toc46204481)

[testh4.txt 49](#_Toc46204482)

[testh5.txt 49](#_Toc46204483)

[testh6.txt 49](#_Toc46204484)

[testh7.txt 50](#_Toc46204485)

[testh8.txt 50](#_Toc46204486)

[testh9.txt 50](#_Toc46204487)

## Conception Détaillé.

## Module Liste (liste\_dyn.ads)

Ce module permet de gérer des listes d’objets abstraits. La liste est faite avec une liste chainé dynamique. Chaque élément du tableau contient un nœud constitué du type abstrait et un pointeur sur le prochain nœud de la liste. Le client doit spécifier lors de l’instanciation le type de la clé, une fonction qui retourne la clé pour un enregistrement et une fonction qui permet de comparer les deux clés.

Services :

### Creer\_Liste

Ce service initialise une liste à l’état vide. Si la liste contient déjà des éléments, elle vas être intialisée à nouveau.

Paramétre : (Entrée-Sortie) Liste : Liste à être initialisé.

Algorithme :

Début.

Si liste.Tête <> NULL alors

Détruire Liste.

Sinon

Allouer un nœud.

Initialiser le pointeur suivant du nœud à NULL.

Tête liste := nœud.

Remorque liste := nœud.

Mettre nombre d’élément à zéro.

Fin.

### Liste\_Vide

Ce service permett de déterminer si une liste est vide. La liste doit avoir été initialisé au préalable.

Paramètre : (Entrée) Liste : Liste qui va être vérifiée si elle est vide.

Retourne : Vrai si vide, sinon faux.

Algorithme :

Début.

Si Nombre Element > 0 alors

Retourne faux.

Sinon

Retourne vrai.

Fin si.

Fin.

### Detruire\_liste

Ce service détruit la liste.

Paramètre : (Entrée-Sortie) Liste : Liste à détruire

Algorithme :

Début.

Ptr := Liste.Tête.

Tant que Ptr <> NULL faire

TmpPtr := Ptr.

Liberer Ptr.

Ptr := TmpPtr.suivant.

Fin tant que.

Mettre nombre d’élément à zéro.

Liste.Remorque := NULL.

Fin.

### Nombre\_Elements

Ce service retourne le nombre d’éléments contenus dans la liste.

Paramètre : (Entrée) Liste : Liste qui vas server à retourner la nombre d’éléments contenus dans elle même.

Retourne : Le nombre d’éléments contenus dans la liste.

Algorithme :

Début.

Retourne Nombre D’éléments.

Fin.

### Inserer\_en\_Ordre

Ce servce insère un élément dans la liste dans l’ordre.

Paramètre : (Entrée-Sortie) Liste : Liste qui vas contenis le nouvel élément.

(Entrée) Elément : Élément à insérer dans la liste.

(Sortie) Booléen : Contient vrais si l’opération a été un succés, sinon faux.

Algorithme :

Début.

Ptr := Liste.tête.

Trouve := faux.

Tant que ( Ptr <> Liste.Remorque) et (Non trouvé) faire.

Si Ptr.élément.clé > Élément.clé alors

Trouvé := vrai.

Else

Ptr := Ptr.suivant.

Fin si.

Fin tant que.

Noeud := Allouer.

Noeud.suivant := NULL.

Si non trouvé alors

Liste.Remorque.suivant := noeud.

Liste.Remorque.Élément := Élément.

Liste.Remorque := Noeud.

Sinon

Noeud.élément := Ptr. élément.

Noeud.suivant := Ptr.suivant.

Ptr.élément := Élément.

Ptr.suivant := Noeud.

Fin.

### Modifier

Ce service permet de modifier un élément de la liste. Il faut que l’élément modifié ne modifie pas la clé.

Paramêtre : (Entrée-Sortie) Liste : La liste à être modifié.

(Entrée) Élément : Élément à être modifié.

(Sortie) Succes : Vrai si l’opération a fonctionné, sinon faux.

Algorithme :

Début.

Succes := faux.

Ptr := Liste.tête.

Tant que ( Ptr <> Liste.Remorque) et (Non trouvé) faire.

Si Ptr.élément.clé = Élément.clé alors

Trouvé := vrai.

Else

Ptr := Ptr.suivant.

Fin si.

Fin tant que.

Si Trouvé alors

Ptr.élément := Élément.

Succes := vrai.

Fin si.

Fin.

### Obtenir

Ce service permet de retrouver un élément dans liste à partir de la clé.

Paramétre : (Entrée) Liste : La liste qui contient l’élément.

(Entrée) Clé : La clé de l’élément.

(Sortie) Élément : L’élément trouvé.

(Sortie) Booléen : Vrai si l’élément a été trouvé, sinon faux.

Algorithme :

Début.

Succes := faux.

Ptr := Liste.tête.

Tant que ( Ptr <> Liste.Remorque) et (Non succes) faire.

Si Ptr.élément.clé = Clé alors

Succes := vrai.

Élément := Ptr.élément.

Sinon

Ptr := Ptr.suivant.

Fin si.

Fin tant que.

Fin.

### Retirer

Ce service enlève un élément de la liste.

Paramètre : (Entrée-Sortie) Liste : La liste qui contient l’élément à être enlever.

(Entrée) Clé : Clé de l’élément à être enelver.

(Sortie) Booléen : Vrai si l’élément a été enlevé, sinon faux.

Algorithme :

Début.

Succes := faux.

Ptr := Liste.tête.

Tant que ( Ptr <> Liste.Remorque) et (Non succes) faire.

Si Ptr.élément.clé = Clé alors

Succes := vrai.

PtrSuivant := Ptr.suivant.

Ptr.Élément := PtrSuivant.Élément.

Ptr.suivant := PtrSuivant.suivant.

Si PtrSuivant = Liste.Remorque

PtrSuivant.suivant := NULL.

Ptr.Suivant := PtrSuivant.

Liste.Remorque := PtrSuivant.

Sinon

Liberer PtrSuivant.

Fin si.

Sinon

Ptr := Ptr.suivant.

Fin si.

Fin tant que.

Fin.

#### Premier

Ce service retourne la clé du premier élément.

Paramètre : (Entrée) Liste : Liste qui contient le premier élément à être retrouvé.

Retourne : La clé de l’élément.

Algorithme :

Début.

Si (NombreElement > 0) alors

Retourne Liste.Tête.clé

Sinon

Lance exception.

Fin.

### Dernier

Ce service retourne la clé du dernier élément.

Paramétre : (Entrée) Liste : Liste qui contient le dernier élément à être retrouvé.

Retourne : La clé de l’élément.

Algorithme :

Début.

Si (NombreÉlément > 0) alors

Ptr := Liste.tête.

Tant que ( Ptr.suivant <> Liste.Remorque) faire.

Ptr := Ptr.suivant.

Fin tant que.

Retourne Ptr.clé.

Sinon

Lance exception.

Fin si.

Fin.

### Successeur

Ce service retourne la clé de l’élément suivant.

Paramétre : (Entrée) Liste : La liste qui contient les éléments.

(Entrée) Clé : Clé de l’élément existant.

(Sortie) CléTrouvé : Clé de l’élément suivant l’élément spécifié.

(Sortie) Booléen : Vrai si il y a un élément suivant, sinon faux.

Algorithme :

Début.

Succes := faux.

Si (NombreÉlément > 0) alors

Ptr := Liste.tête.

Tant que ( Ptr.suivant <> Liste.Remorque) et (Non succes) faire.

Si Ptr.Clé = Clé alors

Succes := vrai.

CléTrouvé := Ptr.Suivant.Clé.

Fin si.

Ptr := Ptr.suivant.

Fin tant que.

Fin si.

Fin.

### Predecesseur

Ce service permet de trouver l’élément avant l’élément spécifié.

Paramétre : (Entrée) Liste : Liste qui contient les éléments.

(Entrée) Clé : Liste de l’élément à trouver.

(Sortie) CléTrouvé : Clé de l’élément précédant l’élément spécifié.

(Sortie) Success : Vrai s’il y a un élément précédent, sinon faux.

Algorithme :

Début.

Succes := faux.

Si (NombreÉlément > 0) alors

Ptr := Liste.tête.

Tant que ( Ptr.suivant <> Liste.Remorque) et (Non succes) faire.

Si Ptr.Suivant.Clé = Clé alors

Succes := vrai.

CléTrouvé := Ptr.Clé.

Fin si.

Ptr := Ptr.suivant.

Fin tant que.

Fin si.

Fin.

### Sauvegarder

Ce service sauvegarde la liste dans un fichier binaire.

Paramétre : (Entrée) Liste : La liste à sauvegarder.

(Entrée) Nom : Nom du fichier à être sauvegarder.

(Sortie) Booléen : Vrai si la liste a été sauvegarder, sinon faux.

Algorithme :

Début.

Créer Fichier.

Ptr := Liste.Tête.

Tant que ( Ptr <> Liste.Remorque) faire.

Ecrire Élément.

Ptr := Ptr.Suivant.

Fin tant que.

Fermer fichier.

Fin.

### Recuperer

Ce service permet de récupérer une liste sauvgardée sur un fichier.

Paramétre : (Entrée) Liste : La liste qui vas contenir ce qu’il y a dans dans le fichier.

(Entrée) Nom : Nom du fichier qui contient la liste.

(Sortie) Booléen : Vrai si le fichier a été lu dans la liste, sinon faux.

Algorithme :

Début.

Ouvrir Fichier.

Allouer Nœud.

Liste.Tete := Nœud.

Tant que (non fin fichier) faire.

Lire Élément.

Nœud.Elément

Nœud.Suivant := Allouer nœud.

Nœud := nœud.suivant.

Fin tant que.

Liste.Remorque := nœud.

Fermer fichier.

Fin.

## Annexe A – Listage des sources.

### TP2.ADA

------------------------------------------------------------------------------

--$Id: tp2.ada,v 1.3 2003/07/16 17:30:13 yann Exp $

------------------------------------------------------------------------------

-- Programme de gestion de cours et d'etudiant.

--

-- Auteur: Yann Bourdeau

-- Courriel: bourdeau.yann@courrier.uqam.ca

-- cours: INF2110-10

------------------------------------------------------------------------------

------------------------------------------------------------------------------

-- Liste des modules utilises.

------------------------------------------------------------------------------

with Ada.Text\_IO; use Ada.text\_io;

with ada.characters.Handling;

with Ada.Strings.Fixed;

with ada.Strings.Maps.Constants;

WITH lecture;

WITH Listes\_tab;

WITH Liste\_Dyn;

------------------------------------------------------------------------------

-- Procedure principal du programme.

-- NIVEAU : 1

------------------------------------------------------------------------------

procedure GestionUQAM is

---------------------------------------------------------------------------

-- Definition des constantes

---------------------------------------------------------------------------

MAX\_LONG\_COURS\_CODE : constant := 7; -- Longueur du code cours.

MAX\_LONG\_COURS\_HORAIRE: constant:= 20; -- Longueur maximum de la chaine

-- de l'horaire.

MAX\_LONG\_ETUDIANT\_MATRICULE: constant:= 4; -- longueur maximum de la

-- matricule.

MAX\_LONG\_ETUDIANT\_PRENOM: constant:= 30; -- Longueur maximum du prenom de

-- l'etudiant.

MAX\_LONG\_ETUDIANT\_NOM: constant:= 30; -- Longueur maximum du nom de

-- l'etudiant.

MAX\_INSCRIPTION\_PAR\_ETUDIANT: constant:= 25; -- Maximum de cours inscrit

-- par etudiant.

---------------------------------------------------------------------------

-- Definition des types.

---------------------------------------------------------------------------

-- Enumeration de la session (ete, hiver, automne)

type Session\_Cours is (hiver, ete, automne);

-- Enumeration de l'etat du cours

TYPE Etat\_Cours is (actif, annule);

-- Enumeration du lieu du cours

TYPE Lieu\_Cours is (Montreal, Quebec, Toronto);

-- Enumeration des notes du cours.

TYPE Note\_Cours is (A,B,C,D,E,Z);

-- Definition du type codeCours

TYPE Code\_Cours is new String ( 1 .. MAX\_LONG\_COURS\_CODE);

-- Definition du type matricule etudiant

TYPE Matricule\_Etudiant is new String( 1.. MAX\_LONG\_ETUDIANT\_MATRICULE);

-- Cle de l'enregistrement cours.

TYPE Cle\_Cours is

record

Code: Code\_Cours; -- Code du cours

Session: Session\_Cours; -- Session du cours

end record;

-- L'enregistrement contient la definition d'une inscription

TYPE EnregistrementInscription is

record

CodeCours : Code\_Cours;

-- Contient le code du cours.

Session : Session\_Cours;-- Contient la session du cours.

Note : Note\_Cours; -- Contient la note du cours.

Statut : Etat\_Cours; -- Contient le statut du cours.

end record;

---------------------------------------------------------------------------

-- Fonction qui compare deux cle, pour savoir si la premiere est plus petite

-- Miveau : Function "callback" pour module de liste.

-- PARAMETRE: Gauche: (in) Cle qui devrais etre inferieur.

-- Droite: (in) Cle qui devrais etre superieur.

-- Retourne : VRAI => Si la cle de gauche est inferieur.

-- FAUX => Si la cle de droite est superieur.

---------------------------------------------------------------------------

FUNCTION CoursPlusPetit (Gauche,Droite: Cle\_Cours) return boolean is

begin

if (Gauche.Code< Droite.Code) then

RETURN true;

elsif (Gauche.Code = Droite.Code) then

if (Gauche.Session< Droite.Session) then

RETURN TRUE;

ELSE

RETURN FALSE;

end if;

else

RETURN false;

end if;

end CoursPlusPetit;

--------------------------------------------------------------------------

-- fonction qui retourne la cle d'un enregistrement de type cours.

-- Miveau : Function "callback" pour module de liste.

-- PARAMETRE: Item: (in) enregistrement du cours.

-- RETOUNE: La Cle.

--------------------------------------------------------------------------

function RetourneCleInscription(Item: EnregistrementInscription) return Cle\_Cours is

-- Declaration des variables.

Cle: Cle\_Cours; -- Cle du cours

begin

Cle.Code := Item.CodeCours;

Cle.Session := Item.Session;

RETURN Cle;

end RetourneCleInscription;

-- Instanciation de module generique.

-- Pour la liste des inscription.

PACKAGE Liste\_Inscription is new Listes\_tab(100,

EnregistrementInscription,

Cle\_Cours,

RetourneCleInscription,

CoursPlusPetit);

-- L'enregistrement contient la definition d'un cours

TYPE EnregistrementCours is

record

CodeCours : Code\_Cours;

-- Contient le code du cours

Session : Session\_Cours;-- Contient la session du cours

Etat : Etat\_Cours; -- Contient si le cours est actif ou annule.

MaxPlace : Positive; -- Nombre maximum de place disponible dans le cours

NumInscrits: natural; -- Nombre d'inscription au cours.

HoraireCours: String (1 .. MAX\_LONG\_COURS\_HORAIRE);

-- L'horaire du cours.

Lieu : Lieu\_Cours; -- Lieu du cours.

--Inscription: Liste; -- Liste des inscriptions au cours.

end record;

-- L'enregistrement contient la definition d'un etudiant.

TYPE EnregistrementEtudiant is

record

Matricule : Matricule\_Etudiant;

-- Matricule de l'etudiant.

Prenom : String (1.. MAX\_LONG\_ETUDIANT\_PRENOM);

-- Prenom de l'etudiant.

Nom : String (1.. MAX\_LONG\_ETUDIANT\_NOM);

-- Nom de l'etudiant.

Inscription : Liste\_Inscription.Liste;

-- Tableau des inscription.

end record;

--------------------------------------------------------------------------

-- fonction qui retourne la cle d'un enregistrement de type cours.

-- Miveau : Function "callback" pour module de liste.

-- PARAMETRE: Item: (in) enregistrement du cours.

-- RETOUNE: La Cle.

--------------------------------------------------------------------------

function RetourneCleCours(Item: EnregistrementCours) return Cle\_Cours is

-- Declaration des variables.

Cle: Cle\_Cours; -- Cle du cours

begin

Cle.Code := Item.CodeCours;

Cle.Session := Item.Session;

RETURN Cle;

end RetourneCleCours;

--------------------------------------------------------------------------

-- fonction qui retourne la cle d'un enregistrement de type etudiant.

-- Miveau : Function "callback" pour module de liste.

-- PARAMETRE: Item: (in) enregistrement de l'etudiant.

-- RETOUNE: La Cle.

--------------------------------------------------------------------------

function RetourneCleEtudiant(Item: EnregistrementEtudiant) return Matricule\_Etudiant is

begin

RETURN Item.Matricule;

end RetourneCleEtudiant;

---------------------------------------------------------------------------

-- Fonction qui compare deux cle, pour savoir si la premiere est plus petite

-- Miveau : Function "callback" pour module de liste.

-- PARAMETRE: Gauche: (in) Cle qui devrais etre inferieur.

-- Droite: (in) Cle qui devrais etre superieur.

-- Retourne : VRAI => Si la cle de gauche est inferieur.

-- FAUX => Si la cle de droite est superieur.

---------------------------------------------------------------------------

FUNCTION EtudiantPlusPetit (Gauche,Droite: Matricule\_Etudiant) return boolean is

begin

if (Gauche<Droite) then

RETURN true;

end if;

RETURN false;

end EtudiantPlusPetit;

---------------------------------------------------------------------------

-- Instanciation des modules generiques

---------------------------------------------------------------------------

-- Pour la liste des cours.

PACKAGE Liste\_Cours is new Liste\_Dyn(EnregistrementCours,

cle\_Cours,

RetourneCleCours,

CoursPlusPetit);

-- Pour la liste des etudiants.

PACKAGE Liste\_Etudiant is new Liste\_Dyn(EnregistrementEtudiant,

Matricule\_Etudiant,

RetourneCleEtudiant,

EtudiantPlusPetit);

--- Entrees/Sorties des valeurs de types énumératifs ------

package ES\_Enum\_SessionCours is new Enumeration\_IO(Session\_Cours);

package ES\_Enum\_LieuCours is new Enumeration\_IO(Lieu\_Cours);

package ES\_Enum\_EtatCours is new Enumeration\_IO(Etat\_Cours);

PACKAGE ES\_Enum\_NoteCours is new Enumeration\_IO(Note\_Cours );

---------------------------------------------------------------------------

-- Fonction qui retourne un code cours a partir d'une chaine

-- NIVEAU 6:

-- PARAMETRE: Chaine: (IN) Code du cours en format STRING.

-- RETOURNE: Le code du cours en format Code\_Cours

---------------------------------------------------------------------------

FUNCTION TransformeChaineEnCodeCours(ChaineCodeCours: in String) return Code\_Cours is

-- Declaration des variables

CodeCours : Code\_Cours; -- Code cours a retourner.

begin

FOR i in ChaineCodeCours'Range loop

CodeCours(i):=ChaineCodeCours(i);

end loop;

RETURN CodeCours;

end TransformeChaineEnCodeCours;

---------------------------------------------------------------------------

-- Fonction qui retourne un matricule a partir d'une chaine

-- NIVEAU 6:

-- PARAMETRE: Chaine: (IN) Matricule etudiant en format STRING.

-- RETOURNE: Le matricule etudiant en format Matricule\_etudiant

---------------------------------------------------------------------------

FUNCTION TransformeChaineEnMatriculeEtudiant(ChaineMatricule: in String) return Matricule\_Etudiant is

-- Declaration des variables

Matricule : Matricule\_Etudiant; -- Matricule Etudiant a retourner.

begin

FOR i in ChaineMatricule'Range loop

Matricule(i):=ChaineMatricule(i);

end loop;

RETURN Matricule;

end TransformeChaineEnMatriculeEtudiant;

---------------------------------------------------------------------------

-- Fonction qui retourne une chaine a partir d'un code cours.

-- NIVEAU 6:

-- PARAMETRE: CodeCours: (IN) Code du cours.

-- RETOURNE: Le code du cours en format String

---------------------------------------------------------------------------

FUNCTION TransformeCodeCoursEnChaine(CodeCours: Code\_cours) return String is

-- Declaration des variables

ChaineCodeCours : String(1..MAX\_LONG\_COURS\_CODE); -- chaine Code cours a retourner.

begin

FOR i in ChaineCodeCours'Range loop

ChaineCodeCours(i):=CodeCours(i);

end loop;

RETURN ChaineCodeCours;

end TransformeCodeCoursEnChaine;

---------------------------------------------------------------------------

-- Fonction qui retourne une chaine a partir d'un matricule etudiant.

-- NIVEAU 6:

-- PARAMETRE: Matricule: (IN) Matricule Etudiant.

-- RETOURNE: Le Matricule en format String

---------------------------------------------------------------------------

FUNCTION TransformeMatriculeEnChaine(Matricule: Matricule\_Etudiant) return String is

-- Declaration des variables

ChaineMatricule: String(1..MAX\_LONG\_ETUDIANT\_MATRICULE); -- chaine Matricule a retourner.

begin

FOR i in ChaineMatricule'Range loop

ChaineMatricule(i):=MAtricule(i);

end loop;

RETURN ChaineMatricule;

end TransformeMatriculeEnChaine;

---------------------------------------------------------------------------

-- Fonction qui lis une session au clavier

-- NIVEAU 6:

-- PARAMETRE: Question: (IN) La question poser a l'utilisateur.

-- RETOURNE: La session du cours.

---------------------------------------------------------------------------

FUNCTION LireSession(Question: in String) return Session\_Cours is

--Declaration des variables.

Session: Session\_Cours:=ete; -- Session lus au clavier

car: character; -- Contient le choix entrer par l'utilisateur.

begin

car := lecture.LireCaractere("123","Session du cours (1-Ete, 2-hiver, 3-automne):");

CASE car IS

WHEN '1' =>

Session:= ete;

WHEN '2' =>

Session:= hiver;

WHEN '3' =>

Session := automne;

WHEN others =>

null;

END CASE;

RETURN Session;

end LireSession;

---------------------------------------------------------------------------

-- Fonction qui lis le lieu d'un cours au clavier.

-- NIVEAU 6:

-- PARAMETRE: Question: (IN) La question poser a l'utilisateur.

-- RETOURNE: La lieu du cours.

---------------------------------------------------------------------------

FUNCTION LireLieu(Question: in String) return Lieu\_Cours is

--Declaration des variables.

Lieu: Lieu\_Cours:=Montreal; -- Lieu lus au clavier

car: character; -- Contient le choix entrer par l'utilisateur.

begin

car := lecture.LireCaractere("123","Lieu du cours (1-Montreal, 2-Quebec, 3-Toronto):");

CASE car IS

WHEN '1' =>

Lieu:=Montreal;

WHEN '2' =>

Lieu:=Quebec;

WHEN '3' =>

Lieu:=Toronto;

WHEN others =>

null;

END CASE;

RETURN Lieu;

end LireLieu;

---------------------------------------------------------------------------

-- Procedure qui trouve l'etudiant a partir du matricule lus au clavier.

-- NIVEAU: 6

-- PARAMETRE: ListeEtudiant: (IN) La liste d'etudiants.

-- Etudiant: (out) L'etudiant trouver.

-- Quitter: (out) VRAI => Transaction annuler par l'utilisateur.

-- FAUX => L'etudiant a ete trouve.

---------------------------------------------------------------------------

procedure LireEtudiantExistant(ListeEtudiant: in Liste\_etudiant.liste;

Etudiant: out EnregistrementEtudiant;

Quitter: out boolean) is

-- Declaration des variables.

succes: boolean; -- Determine si la fonction a reussis.

Car: Character; -- Reponse lus au clavier pour savoir si l'utilisateur veut continuer.

begin

quitter:=false;

-- Lis matricule etudiant au clavier

loop

Etudiant.Matricule := TransformeChaineEnMatriculeEtudiant(Ada.Strings.Fixed.Translate(

Lecture.LireChaine(" ","Entrez le matricule etudiant:"),

Ada.Strings.Maps.Constants.Upper\_Case\_Map));

-- Retrouve l'etudiant

Liste\_etudiant.Obtenir (ListeEtudiant,Etudiant.Matricule,Etudiant,succes);

IF not succes THEN

put\_line("Le matricule n'existe pas.");

Car := ada.characters.handling.To\_Upper(Lecture.LireCaractere("OoNn","Continuer (O/N):"));

if(Car = 'N') then

quitter := true;

end if;

END IF;

EXIT when succes or quitter;

end loop;

end LireEtudiantExistant;

---------------------------------------------------------------------------

-- Procedure qui trouve un cours a partir de la cle cours lus au clavier.

-- NIVEAU: 6

-- PARAMETRE: ListeCours: (IN) La liste des cours.

-- Cours: (out) Le cours trouve.

-- Quitter: (out) VRAI => Transaction annuler par l'utilisateur.

-- FAUX => Le cours a ete trouve.

---------------------------------------------------------------------------

PROCEDURE LireCoursExistant(ListeCours: in Liste\_cours.liste;

Cours: out EnregistrementCours;

Quitter: out boolean ) is

-- Declaration des variables temporaire.

CleCours: Cle\_Cours; -- Contient la cle du cours demander

succes: boolean; -- Determine si la fonction a reussis ou pas.

Car: Character; -- Reponse lus au clavier pour savoir si l'utilisateur veut continuer.

begin

quitter:= false;

loop

-- lis la session.

CleCours.Session := LireSession("Session du cours(ete, hiver, automne):");

-- Lis le cours.

CleCours.Code := TransformeChaineEnCodeCours(

Ada.Strings.Fixed.Translate(

Lecture.LireChaine("AAANNNN","Entrez le code cours (AAANNNN):"),

Ada.Strings.Maps.Constants.Upper\_Case\_Map));

-- Valide si le cours existe.

Liste\_Cours.Obtenir(ListeCours,CleCours,Cours,succes);

IF not succes THEN

put\_line("Le cours n'existe pas.");

Car := ada.characters.handling.To\_Upper(Lecture.LireCaractere("OoNn","Continuer (O/N):"));

if(Car = 'N') then

quitter := true;

end if;

END IF;

EXIT when succes or quitter;

end loop;

end LireCoursExistant;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction de creer un nouveau cours.

-- Gestion cours

-- NIVEAU: 5

-- PARAMETRE: ListeCour: (IN OUT) Liste des cours.

---------------------------------------------------------------------------

procedure EffectuerTransactionCreerNouveauCours(ListeCours: in out Liste\_cours.liste) is

-- Declaration des variables

Cours: EnregistrementCours; -- Contient le cours creer.

TempCours: EnregistrementCours; -- Enregistrement temporaire qui sert a verifier si

-- le cours existe deja.

succes: boolean; -- Determine si la procedure a reussis.

quitter: boolean := false; -- Determine si l'utilisateur veut continuer en cas d'erreur.

Car: Character; -- Reponse lus au clavier pour savoir si l'utilisateur veut continuer.

CoursAnnule: boolean := false; -- Determine si le cours a ete annule.

begin

put\_line("Creer un nouveau cours.");

put\_line("-----------------------");

loop

-- Lis le cours au clavier

Cours.CodeCours := TransformeChaineEnCodeCours(Ada.Strings.Fixed.Translate(Lecture.LireChaine("AAANNNN","Entrez le code cours (AAANNNN):"),Ada.Strings.Maps.Constants.Upper\_Case\_Map));

-- Lis la session

Cours.Session:=LireSession("Session du cours(ete, hiver, automne):");

--Verifie si le cours existe deja

Liste\_cours.Obtenir (ListeCours,RetourneCleCours(Cours),TempCours,Succes);

-- Verifie si le cours n'as pas deja ete annule!

IF succes THEN

IF TempCours.Etat = annule THEN

CoursAnnule := true;

succes := false;

END IF;

END IF;

-- Quitte si le cours n'est pas trouve

EXIT when not succes;

-- Demande a l'usager si il veut continuer.

put\_line("Le cours existe deja.");

new\_line;

Car := ada.characters.handling.To\_Upper(Lecture.LireCaractere("OoNn","Continuer (O/N):"));

if(Car = 'N') then

quitter := true;

end if;

-- Termine la transaction si l'utilisateur ne veut plus entrer le cours.

EXIT when quitter;

end loop;

IF not quitter THEN

Cours.Etat:=actif;

Cours.MaxPlace:=Lecture.LirePositive("Nombre de place disponible:");

Cours.NumInscrits := 0;

Cours.HoraireCours:=Lecture.LireChaine(" ","Entrez l'horaire du cours (format libre):");

Cours.Lieu := LireLieu("Lieu du cours(Montreal, Quebec, Toronto):");

IF NOT CoursAnnule THEN

-- Insere le cours dans la liste.

Liste\_cours.Inserer\_en\_Ordre(ListeCours,Cours,Succes);

ELSE

-- Si le cours existe deja pcq annule, le remplacer.

Liste\_Cours.Modifier(ListeCours,Cours,Succes);

END IF;

else

succes:=false;

end if;

IF NOT quitter THEN

IF succes THEN

put\_line("Le cours a ete ajouter.");

else

put\_line("ERREUR: Impossible d'ajouter le cours.");

END IF;

Lecture.AttendreTouche;

END IF;

end EffectuerTransactionCreerNouveauCours;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction d'annuler un cours

-- Gestion cours

-- NIVEAU: 5

-- PARAMETRE: ListeEtudiant: (IN OUT) Liste des etudiants.

-- ListeCours: (IN OUT) Liste des cours.

---------------------------------------------------------------------------

procedure EffectuerTransactionAnnulerCours(ListeCours: in out Liste\_cours.liste;

ListeEtudiant: in out Liste\_etudiant.liste) is

-- Declaration des variables.

Cours: EnregistrementCours; -- Contient l'enregistrement du cours.

Etudiant: EnregistrementEtudiant; -- Contient l'etudiant.

Inscription: EnregistrementInscription; -- Contient l'inscription.

Quitter: Boolean; -- Determine si l'utilisateur veux terminer la transaction.

car: character; -- Contient la reponse de l'utilisateur.

Succes: Boolean := true; --Determine si la derniere fonction a reussis.

begin

put\_line("Annuler cours.");

put\_line("--------------");

LireCoursExistant(ListeCours,Cours,Quitter);

IF NOT quitter THEN

Car := ada.characters.handling.To\_Upper(Lecture.LireCaractere("OoNn","Annulation du cours (O/N):"));

IF (Car = 'O') THEN

Cours.Etat:=annule;

IF Cours.NumInscrits>0 THEN

-- Mets le nombre d'inscription a 0.

Cours.NumInscrits := 0;

--Mets a jour le cours.

Liste\_cours.Modifier (ListeCours,Cours,Succes);

IF not succes THEN

put\_line("ERREUR: Impossible de mettre a jour le cours.");

END IF;

--enleve les etudiants inscrits.

Etudiant.Matricule:=Liste\_etudiant.Premier(ListeEtudiant);

loop

Liste\_Etudiant.Obtenir(ListeEtudiant,Etudiant.Matricule,Etudiant,succes);

IF succes THEN

-- Verifie si il est inscrit au cours.

Liste\_Inscription.Obtenir(Etudiant.Inscription,RetourneCleCours(Cours),Inscription,succes);

IF succes THEN

-- Modifie l'inscription

Inscription.Statut:=annule;

Liste\_Inscription.Modifier(Etudiant.Inscription,Inscription,succes);

IF succes THEN

-- Modifie l'etudiant.

Liste\_Etudiant.Modifier(ListeEtudiant,Etudiant,succes);

IF NOT succes THEN

put\_line("ERREUR: Impossible de modifier l'etudiant.");

END IF;

ELSE

put\_line("ERREUR: Impossible de modifier l'inscription.");

END IF;

put\_line("Le cours a ete annule.");

END IF;

END IF;

-- Passe a l'etudiant suivant.

Liste\_Etudiant.Successeur (ListeEtudiant,Etudiant.Matricule,Etudiant.Matricule,Succes);

-- Quitte si aucune autre etudiant.

EXIT WHEN NOT succes;

end loop;

ELSE

-- personne d'inscrit, enleve le cours.

Liste\_Cours.Retirer(ListeCours,RetourneCleCours(Cours),Succes);

IF NOT succes THEN

put\_line("ERREUR: Impossible d'annuler le cours.");

ELSE

put\_line("Le cours a ete annule.");

END IF;

END IF;

END IF;

END IF;

IF NOT quitter THEN

Lecture.AttendreTouche;

END IF;

end EffectuerTransactionAnnulerCours;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction de lister les cours

-- Gestion cours

-- NIVEAU: 5

-- PARAMETRE: ListeCour: (IN) Liste des cours.

---------------------------------------------------------------------------

procedure EffectuerTransactionListerLesCours(ListeCours: in Liste\_cours.liste) is

-- Declaration des variables.

Cle : Cle\_Cours; -- Cle (code\_cours) de l'enregistrement courant.

Cours: EnregistrementCours; -- Enregistrement du cours.

Succes: Boolean := true; --Determine si la derniere fonction a reussis.

Espace: String(1..30) :=" ";

-- Contient le "padding" entre champs.

begin

put\_line("Liste des cours");

put\_line("Code Session Etat Lieu Nombre de place Inscription");

put\_line("===============================================================================");

if ( not Liste\_Cours.Liste\_Vide ( ListeCours)) then

Cle:=Liste\_Cours.Premier (ListeCours);

-- Enumere tous les cours.

loop

Liste\_Cours.Obtenir (ListeCours,Cle,Cours,Succes);

IF not succes THEN

put\_line("ERREUR: Impossible d'obtenir le cours.");

END IF;

IF Succes THEN

-- Affiche le cours

put(TransformeCodeCoursEnChaine(Cours.CodeCours) & " ");

ES\_Enum\_EtatCours.put(Cours.Etat);

-- Mets le nombre d'espace blanc necessaire entre

-- les deux champs

put(Espace(1..10-Etat\_Cours'image(Cours.etat)'last));

ES\_Enum\_SessionCours.put(Cours.Session);

put(Espace(1..10-Session\_Cours'image(Cours.Session)'last));

ES\_Enum\_LieuCours.put(Cours.Lieu);

put(Espace(1..10-Lieu\_Cours'image(Cours.Lieu)'last));

put(Integer'Image(Cours.MaxPlace));

put(Espace(1..19-Integer'image(Cours.MaxPlace)'last));

put\_line(Integer'Image(Cours.NumInscrits));

Liste\_Cours.Successeur(ListeCours,Cle,Cle,Succes);

IF not succes THEN

if not (Liste\_Cours.Dernier (ListeCours) = Cle) then

put\_line("ERREUR: Impossible de trouver l'enregistrement suivant.");

end if;

END IF;

END IF;

EXIT when not succes;

end loop;

else

put\_line("Aucun cours disponible.");

end if;

Lecture.AttendreTouche;

end EffectuerTransactionListerLesCours;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction de lister les inscriptions

-- Gestion cours

-- NIVEAU: 5

-- PARAMETRE: ListeCours: (IN ) Liste des cours.

-- ListeEtudiant: (IN ) Liste des etudiants.

---------------------------------------------------------------------------

procedure EffectuerTransactionListerLesInscriptions(ListeCours: in Liste\_cours.liste;

ListeEtudiant: in Liste\_etudiant.liste) is

-- Declaration des variables.

Cours: EnregistrementCours; -- Contient le cours.

Etudiant: EnregistrementEtudiant; -- Contient l'etudiant.

Inscription: EnregistrementInscription; -- Contient l'inscription.

Quitter: boolean; -- Determine si l'utilisateur veut annuler la transaction.

InscriptionTrouver: boolean:=false; -- Indique si au moins une inscription a ete trouver.

succes: boolean; --Indique si la function a reussis.

begin

LireCoursExistant(ListeCours,Cours,Quitter);

IF NOT quitter THEN

put\_line("Liste des inscriptions.");

put\_line("Matricule Prenom Nom ");

put\_line("===============================================================================");

-- Determine si la liste des etudiant est vide

if not Liste\_Etudiant.Liste\_Vide(ListeEtudiant) then

-- Trouve le premier etudiant.

Etudiant.Matricule:= Liste\_Etudiant.Premier(ListeEtudiant);

loop

-- Obtient l'etudiant

Liste\_Etudiant.Obtenir(ListeEtudiant,Etudiant.Matricule,Etudiant,succes);

IF succes THEN

-- Trouve l'inscription au cours.

Liste\_Inscription.Obtenir(Etudiant.Inscription,RetourneCleCours(Cours),Inscription,succes);

IF succes and Inscription.statut = actif THEN

-- l'inscription existe.

InscriptionTrouver:=true;

-- Affiche l'inscription

put(TransformeMatriculeEnChaine(Etudiant.Matricule) );

put(" ");

put(Etudiant.prenom);

put(" ");

put\_line(Etudiant.nom);

END IF;

-- Passe au prochain etudiant.

Liste\_Etudiant.Successeur (ListeEtudiant,Etudiant.Matricule,Etudiant.Matricule,Succes);

ELSE

put\_line("ERREUR: Impossible d'obtenir l'etudiant.");

END IF;

EXIT when not succes;

end loop;

IF NOT InscriptionTrouver THEN

put\_line("Aucune inscription.");

END IF;

else

put\_line("Aucune inscription.");

end if;

END IF;

IF NOT quitter THEN

Lecture.AttendreTouche;

END IF;

end EffectuerTransactionListerLesInscriptions;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction de creer un nouveau dossier

-- Gestion etudiants

-- NIVEAU: 5

-- PARAMETRE: ListeEtudiant: (IN OUT) Liste des etudiants.

---------------------------------------------------------------------------

procedure EffectuerTransactionCreerNouveauDossier(ListeEtudiant: in out Liste\_etudiant.liste) is

-- Declacration des variables.

Etudiant: EnregistrementEtudiant; -- L'etudiant creer.

FiltreNom: String(1..Etudiant.Nom'Last); -- Filtre du nom pour la lecture

-- de la chaine

FiltrePrenom: String(1..Etudiant.Prenom'Last); -- Filtre du prenom pour

-- la lecture de la chaine.

MatriculeCreer: boolean:= false; -- Determine si le matricule a ete creer.

succes: boolean; -- Determine si la fonction a reussis.

ChaineId: String(1..10) :="0123456789"; -- Constante qui contient les nombres

-- a ajouter au matricule.

begin

-- Initialise les filtres a des espaces blanc (pour accepter n'importe quelle caracteres).

FOR i in FiltreNom'Range loop

FiltreNom(i):=' ';

end loop;

FOR i in FiltrePrenom'Range loop

FiltrePrenom(i):=' ';

end loop;

put\_line("Creer un nouveau dossier.");

put\_line("-------------------------");

-- Creer la liste des inscriptions.

Liste\_Inscription.Creer\_Liste(Etudiant.Inscription);

-- Lis les deux chaines.

Etudiant.Nom:=Lecture.LireChaine(FiltreNom,"Entrez nom:");

Etudiant.Prenom:=Lecture.LireChaine(FiltrePrenom,"Entrez prenom:");

-- Genere le matricule.

-- Copie les 3 premieres lettres du nom

FOR i in 1..3 loop

Etudiant.Matricule(i):=ada.characters.handling.To\_Upper(Etudiant.nom(i));

end loop;

-- Mets le caracter unique a la fin du nom.

FOR i in 1..10 loop

IF not MatriculeCreer THEN

Etudiant.Matricule(4):= ChaineId(i) ;

-- Essaye d'ajouter l'etudiant.

Liste\_etudiant.Inserer\_en\_Ordre (ListeEtudiant,Etudiant,Succes);

IF succes THEN

MatriculeCreer := true;

END IF;

END IF;

end loop;

IF MatriculeCreer THEN

put("L'etudiant ");

put(TransformeMatriculeEnChaine(Etudiant.Matricule));

put\_line(" a ete ajoute.");

ELSE

-- Detruire la liste d'inscription

Liste\_Inscription.Detruire\_liste(Etudiant.Inscription);

put\_line("ERREUR: Impossible d'ajouter l'etudiant.");

END IF;

Lecture.AttendreTouche;

end EffectuerTransactionCreerNouveauDossier;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction de consulter un dossier

-- Gestion etudiants

-- NIVEAU: 5

-- PARAMETRE: ListeEtudiant: (IN) Liste des etudiants.

---------------------------------------------------------------------------

procedure EffectuerTransactionConsulterUnDossier(ListeEtudiant: in Liste\_etudiant.liste) is

-- Declarations des variables.

Etudiant: EnregistrementEtudiant; -- Contient l'etudiant a consulter.

succes: boolean; -- Determine si la procedure a reussis.

Cle: Cle\_Cours; -- Contient la cle du cours.

Inscription: EnregistrementInscription;-- Contient l'inscription a afficher.

Quitter: Boolean :=false; -- Determine si l'utilisateur veut annuler la transaction.

Espace: String(1..30) :=" ";

-- Contient le "padding" entre champs.

begin

put\_line("Consulter un dossier.");

put\_line("---------------------");

-- Lis le matricule.

LireEtudiantExistant(ListeEtudiant,Etudiant,Quitter);

IF not quitter THEN

-- Trouve l'enregistrement.

Liste\_etudiant.Obtenir (ListeEtudiant,Etudiant.Matricule,Etudiant,Succes);

-- Affiche l'information

IF succes THEN

new\_line(2);

put("Matricule: ");

put\_line(TransformeMatriculeEnChaine(Etudiant.Matricule) );

put("Nom: ");

put\_line(Etudiant.Nom);

put("Prenom: ");

put\_line(Etudiant.Prenom);

put\_line("Inscription");

put\_line("Cours session Statut Note");

put\_line("===============================================================================");

IF not Liste\_Inscription.Liste\_Vide (Etudiant.Inscription) THEN

-- Trouve le premier enregistrement.

Cle:=Liste\_Inscription.Premier(Etudiant.Inscription);

loop

-- Charge l'enregistrement.

Liste\_Inscription.Obtenir (Etudiant.Inscription,Cle,Inscription,succes);

IF succes THEN

-- affiche l'enregistrement

put(TransformeCodeCoursEnChaine(Inscription.CodeCours) & " ");

ES\_Enum\_SessionCours.put(Inscription.Session);

-- Mets le nombres d'espace blanc necessaire entre

-- les deux champs

put(Espace(1..12-Session\_Cours'image(Inscription.Session)'last));

ES\_Enum\_EtatCours.put(Inscription.Statut);

put(Espace(1..15-Etat\_Cours'image(Inscription.Statut)'last));

ES\_Enum\_NoteCours.put(Inscription.Note);

new\_line;

-- Passe au prochaine enregistrement.

Liste\_Inscription.Successeur (Etudiant.Inscription,Cle,Cle,Succes);

ELSE

put\_line("ERREUR: Impossible d'obtenir l'inscription.");

END IF;

EXIT when not succes;

end loop;

ELSE

put\_line("Aucune inscription.");

END IF;

new\_line;

put\_line("La note Z indique un resultat non disponible");

ELSE

put\_line("Impossible d'obtenir le dossier.");

END IF;

END IF;

IF NOT quitter THEN

Lecture.AttendreTouche;

END IF;

end EffectuerTransactionConsulterUnDossier;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction d'ajouter une inscription

-- Gestion etudiants

-- NIVEAU: 5

-- PARAMETRE: ListeEtudiant: (IN OUT) Liste des etudiants.

-- ListeCours: (IN OUT) Liste des cours.

---------------------------------------------------------------------------

procedure EffectuerTransactionAjouterInscription(ListeEtudiant: in out Liste\_etudiant.liste;

ListeCours: in out Liste\_cours.liste) is

-- Declarations des variables.

Etudiant: EnregistrementEtudiant; -- Contient l'etudiant.

Cours: EnregistrementCours; -- Contient le cours.

Inscription: EnregistrementInscription; -- Contient l'inscription.

Quitter: Boolean := false; -- Determine si l'utilisateur annule la transaction.

succes: boolean; -- Determine si une fonction a reussis.

car: character; -- Contient la reponse de l'utilisateur.

begin

put\_line("Ajouter une inscription.");

put\_line("------------------------");

-- Lis le matricule.

LireEtudiantExistant(ListeEtudiant,Etudiant,Quitter);

IF not quitter THEN

LireCoursExistant(ListeCours,Cours,Quitter);

END IF;

IF not quitter THEN

Car := ada.characters.handling.To\_Upper(Lecture.LireCaractere("OoNn","Inscription au cours (O/N):"));

END IF;

IF not quitter and car ='O' THEN

-- Verifie si il y a de la place disponible

IF Cours.NumInscrits + 1 <= Cours.MaxPlace THEN

--Verifie si le cours est actif.

IF Cours.Etat = actif THEN

-- Incremente le nombre d'inscrits.

Cours.NumInscrits := Cours.NumInscrits + 1;

-- Creer l'enregistrement inscription.

Inscription.CodeCours := Cours.CodeCours;

Inscription.Session := Cours.Session;

Inscription.Note := Z;

Inscription.Statut:= actif;

-- Ajoute l'inscription dans la liste d'inscription de l'etudiant

Liste\_Inscription.Inserer\_en\_Ordre(Etudiant.Inscription,Inscription,Succes);

-- Verifie si l'inscription existante est pas annule!

IF NOT succes then

-- Lire l'inscription.

Liste\_Inscription.Obtenir(Etudiant.Inscription,RetourneCleInscription(Inscription),Inscription,Succes);

IF succes THEN

-- Si l'inscription est annule, remettre actif et la sauvegarder.

IF Inscription.statut /= actif THEN

Inscription.statut := actif;

Liste\_Inscription.Modifier (Etudiant.Inscription,Inscription,succes);

ELSE

succes := false;

END IF;

END IF;

END IF;

IF succes THEN

-- Mets a jour l'etudiant.

Liste\_Etudiant.Modifier(ListeEtudiant,Etudiant,succes);

IF succes THEN

-- mets a jours le cours.

Liste\_cours.Modifier (ListeCours,Cours,succes);

IF succes THEN

put\_line("Inscription faite.");

ELSE

put\_line("ERREUR: impossible de modifier le cours.");

END IF;

ELSE

put\_line("ERREUR: Impossible de modifier l'etudiant.");

END IF;

ELSE

put\_line("ERREUR: Impossible d'ajouter l'inscription.");

END IF;

ELSE

put\_line("Cours annule.");

END IF;

ELSE

put\_line("Aucune place disponible.");

END IF;

ELSE

put\_line("L'inscription n'as pas ete faite.");

END IF;

IF Not quitter THEN

Lecture.AttendreTouche;

END IF;

end EffectuerTransactionAjouterInscription;

---------------------------------------------------------------------------

-- Procedure qui effectue la transaction d'annuler une inscription

-- Gestion etudiants

-- NIVEAU: 5

-- PARAMETRE: ListeEtudiant: (IN OUT) Liste des etudiants.

-- ListeCours: (IN OUT) Liste des cours.

---------------------------------------------------------------------------

procedure EffectuerTransactionAnnulerInscription(ListeEtudiant: in out Liste\_etudiant.liste;

ListeCours: in out Liste\_cours.liste) is

-- Declarations des variables

Etudiant: EnregistrementEtudiant; -- Contient l'etudiant.

Cours: EnregistrementCours; --Contient le cours.

Inscription: EnregistrementInscription; -- Contient l'inscription.

Quitter: Boolean := false; -- Determine si l'utilisateur annule la transaction.

succes: boolean; -- Determine si une fonction a reussis.

car: character; -- Contient la reponse de l'utilisateur.

begin

put\_line("Annuler une inscription.");

put\_line("------------------------");

-- Lis le matricule.

LireEtudiantExistant(ListeEtudiant,Etudiant,Quitter);

IF NOT quitter THEN

LireCoursExistant(ListeCours,Cours,Quitter);

END IF;

IF NOT quitter THEN

-- Verifie que l'etudiant est inscrit au cours.

Liste\_Inscription.Obtenir (Etudiant.Inscription,RetourneCleCours(Cours),Inscription,Succes);

IF succes THEN

-- Verifie que le cours est toujours actif.

IF (Inscription.Statut = actif ) THEN

Car := ada.characters.handling.To\_Upper(Lecture.LireCaractere("OoNn","Annulation du cours (O/N):"));

IF (Car = 'O') THEN

-- Diminue le nombre d'inscription au cours.

Inscription.Statut:=annule;

Cours.NumInscrits:=Cours.NumInscrits - 1;

-- Mets a jour l'inscription

Liste\_inscription.Modifier (Etudiant.Inscription,Inscription,succes);

IF succes THEN

-- Mets a jour l'etudiant

Liste\_Etudiant.Modifier(ListeEtudiant,Etudiant,succes);

IF succes THEN

-- mets a jour le cours.

Liste\_cours.Modifier(ListeCours,Cours,succes);

IF succes THEN

put\_line("Cours annuler.");

ELSE

put\_line("ERREUR: Impossible de mettre a jour le cours.");

END IF;

ELSE

put\_line("ERREUR: Impossible de mettre a jour l'etudiant.");

END IF;

ELSE

put\_line("ERREUR: Impossible de mettre a jour l'inscription.");

END IF;

ELSE

put\_line("Transaction annule.");

END IF;

ELSE

put\_line("L'inscription est deja annule.");

END IF;

ELSE

put\_line("Aucune inscription pour ce cours.");

END IF;

END IF;

IF NOT Quitter THEN

Lecture.AttendreTouche;

END IF;

end EffectuerTransactionAnnulerInscription;

---------------------------------------------------------------------------

-- Procedure qui affiche le menu de gestion des cours.

-- NIVEAU: 4

-- Aucun parametre

---------------------------------------------------------------------------

procedure AfficherMenuGestionCours is

begin

new\_line(2);

put\_line("\*\*\*\*\*\*\*\* MENU DE GESTION DES COURS \*\*\*\*\*\*\*\*\*\*");

put\_line("\* C - Creer un nouveau cours. \*");

put\_line("\* A - Annuler un cours. \*");

put\_line("\* L - Lister les cours. \*");

put\_line("\* I - Lister les inscription. \*");

put\_line("\* Q - Quitter le menu de gestion des cours. \*");

put\_line("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

end AfficherMenuGestionCours;

---------------------------------------------------------------------------

-- Procedure qui affiche le menu de gestion des etudiants.

-- NIVEAU: 4

-- Aucun parametre

---------------------------------------------------------------------------

procedure AfficherMenuGestionEtudiant is

begin

new\_line(2);

put\_line("\*\*\*\*\*\*\*\*\* MENU DE GESTION DES ETUDIANTS \*\*\*\*\*\*\*\*\*");

put\_line("\* C - Creer un nouveau dossier. \*");

put\_line("\* O - Consulter un dossier. \*");

put\_line("\* A - Ajouter une inscription. \*");

put\_line("\* U - Annuler une inscription. \*");

put\_line("\* Q - Quitter le menu de gestion des etudiants. \*");

put\_line("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

end AfficherMenuGestionEtudiant;

---------------------------------------------------------------------------

-- Procedure qui gere le menu de cours

-- NIVEAU: 3

-- PARAMETRE: ListeCour: (IN OUT) Liste des cours.

-- ListeEtudiant: (IN OUT) Liste des etudiants.

---------------------------------------------------------------------------

PROCEDURE EffectuerGestionCours(ListeCours: in out Liste\_cours.liste;

ListeEtudiant: in out Liste\_etudiant.liste) is

TransactionCours: character; -- Contient la transaction choisi par

-- l'utilisateur.

begin

loop

AfficherMenuGestionCours;

-- Lire la transaction choisi par l'utilisateur.

TransactionCours:=ada.characters.handling.To\_Upper(lecture.LireCaractere("caliqCALIQ","Choix(C,A,L,I,Q):"));

CASE TransactionCours IS

WHEN 'C' =>

EffectuerTransactionCreerNouveauCours(ListeCours);

WHEN 'A' =>

EffectuerTransactionAnnulerCours(ListeCours,ListeEtudiant);

WHEN 'L' =>

EffectuerTransactionListerLesCours(ListeCours);

WHEN 'I' =>

EffectuerTransactionListerLesInscriptions(ListeCours,ListeEtudiant);

WHEN 'Q' =>

EXIT;

WHEN others =>

null;

END CASE;

end loop;

end EffectuerGestionCours;

---------------------------------------------------------------------------

-- Procedure qui gere le menu des etudiants

-- NIVEAU: 3

-- PARAMETRE: ListeCour: (IN OUT) Liste des cours.

-- ListeEtudiant: (IN OUT) Liste des etudiants.

---------------------------------------------------------------------------

PROCEDURE EffectuerGestionEtudiant(ListeCours: in out Liste\_cours.liste;

ListeEtudiant: in out Liste\_etudiant.liste) is

TransactionEtudiant: character; -- Contient la transaction choisi par

-- l'utilisateur.

begin

loop

AfficherMenuGestionEtudiant;

-- Lire la transaction choisi par l'utilisateur.

TransactionEtudiant:=ada.characters.handling.To\_Upper(lecture.LireCaractere("COAUQcoauq","Choix(C,O,A,U,Q):"));

CASE TransactionEtudiant IS

WHEN 'C' =>

EffectuerTransactionCreerNouveauDossier(ListeEtudiant);

WHEN 'O' =>

EffectuerTransactionConsulterUnDossier(ListeEtudiant);

WHEN 'A' =>

EffectuerTransactionAjouterInscription(ListeEtudiant,ListeCours);

WHEN 'U' =>

EffectuerTransactionAnnulerInscription(ListeEtudiant,ListeCours);

WHEN 'Q' =>

EXIT;

WHEN others =>

null;

END CASE;

end loop;

end EffectuerGestionEtudiant;

---------------------------------------------------------------------------

-- Procedure que affiche le menu principal

-- NIVEAU: 2

-- Aucun parametre.

---------------------------------------------------------------------------

PROCEDURE AfficherMenuPrincipal is

begin

new\_line(2);

put\_line("\*\*\*\*\*\*\* MENU PRINCIPAL \*\*\*\*\*\*\*");

put\_line("\* C - Gestion des cours. \*");

put\_line("\* E - Gestion des etudiants. \*");

put\_line("\* Q - Quitter programme. \*");

put\_line("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

end AfficherMenuPrincipal;

---------------------------------------------------------------------------

-- Procedure que fait l'initialisation du programme

-- 1- Affiche en-tete.

-- 2- Creer les listes.

-- 3- Lis les fichiers des listes.

-- NIVEAU: 2

-- PARAMETRE: ListeCour: (IN OUT) Liste des cours.

-- ListeEtudiant: (IN OUT) Liste des etudiants.

---------------------------------------------------------------------------

PROCEDURE InitialisationProgramme(ListeCour: in out Liste\_cours.liste;

ListeEtudiant: in out Liste\_etudiant.liste) is

-- Declaration des variables.

Succes: boolean; -- Determine si la fonction appeler a reussis.

begin

-- Affiche l'en-tete du programme.

put\_line("Gestion de cours et d'etudiant");

put\_line("Auteur: Yann Bourdeau");

-- Creer les deux listes.

Liste\_cours.Creer\_Liste(ListeCour);

Liste\_etudiant.Creer\_Liste(ListeEtudiant);

-- Charge les listes.

liste\_cours.Recuperer(ListeCour,"cours.bin",Succes);

if ( not Succes) then

put\_line("Impossible de lire cours.bin.");

end if;

liste\_etudiant.recuperer(listeEtudiant,"etudiant.bin",succes);

if(not succes) then

put\_line("Impossible de lire etudiant.bin.");

end if;

end InitialisationProgramme;

---------------------------------------------------------------------------

-- Procedure que fait la termination du programme

-- 1 - Sauvegarde les fichiers des listes.

-- 2 - Detruire les listes.

-- NIVEAU: 2

-- PARAMETRE: ListeCour: (IN OUT) Liste des cours.

-- ListeEtudiant: (IN OUT) Liste des etudiants.

---------------------------------------------------------------------------

PROCEDURE TerminationProgramme(ListeCour: in out Liste\_cours.liste;

ListeEtudiant: in out Liste\_etudiant.liste) is

-- Declaration des variables.

Succes: boolean; -- Determine si la fonction appeler a reussis.

Etudiant: EnregistrementEtudiant; -- Contient l'etudiant obtenus.

SuccesEtudiant: boolean; --Determine si les procedurs de liste etudiantes

-- ont reussis.

begin

liste\_cours.Sauvegarder(ListeCour,"cours.bin",succes);

IF (not succes) THEN

put\_line("Impossible de sauvegarder cours.bin.");

END IF;

liste\_etudiant.sauvegarder(ListeEtudiant,"etudiant.bin",succes);

IF (not succes) THEN

put\_line("Impossible de sauvegarder etudiant.bin");

END IF;

-- Parcoure la liste des etudiants pour liberes les listes

-- d'inscription.

IF not liste\_etudiant.Liste\_Vide(ListeEtudiant) THEN

-- La liste des etudiants n'est pas vide.

-- Trouve le premier

Etudiant.Matricule:=liste\_etudiant.Premier (ListeEtudiant);

loop

-- retrouve l'enregistrement

liste\_etudiant.Obtenir (ListeEtudiant,Etudiant.Matricule,Etudiant,SuccesEtudiant);

if SuccesEtudiant then

-- Detruire la liste

Liste\_Inscription.Detruire\_Liste(Etudiant.Inscription);

-- Passe au prochaine enregistrement

Liste\_Etudiant.Successeur (ListeEtudiant,Etudiant.matricule,Etudiant.Matricule,SuccesEtudiant);

end if;

EXIT when not SuccesEtudiant;

end loop;

END IF;

-- Libere les deux listes.

Liste\_cours.Detruire\_liste(ListeCour);

Liste\_etudiant.Detruire\_liste(ListeEtudiant);

put\_line ("Au revoir!");

end TerminationProgramme;

---------------------------------------------------------------------------

-- Definition des variables.

---------------------------------------------------------------------------

TransactionPrincipal: Character; --Transaction principal entrer par l'utilisateur

ListeCour: Liste\_cours.liste; -- Contient la liste des cours

ListeEtudiant: Liste\_etudiant.liste; -- Contient la liste des etudiants.

begin

InitialisationProgramme(ListeCour,ListeEtudiant);

loop

AfficherMenuPrincipal;

-- lis la transaction choisi par l'utilisateur

TransactionPrincipal:=ada.characters.handling.To\_Upper(lecture.LireCaractere("CEQceq","Choix(C,E,Q):"));

-- Sort de la boucle si l'utilisateur choisis de quitter.

CASE TransactionPrincipal IS

WHEN 'C' =>

EffectuerGestionCours(listeCour,ListeEtudiant);

WHEN 'E' =>

EffectuerGestionEtudiant(ListeCour,ListeEtudiant);

WHEN 'Q' =>

EXIT ;

WHEN others =>

null;

END CASE;

end loop;

TerminationProgramme(ListeCour,ListeEtudiant);

exception

when others => put\_Line("Exception inconnus. Fin du programme.");

end GestionUQAM;

### LECTURE.ADS

------------------------------------------------------------------------------

--$Id: lecture.ads,v 1.1 2003/07/16 01:40:46 yann Exp $

------------------------------------------------------------------------------

-- Module qui fait une gestion evolue des entrees sortie au clavier.

--

-- Auteur: Yann Bourdeau

-- Courriel: bourdeau.yann@courrier.uqam.ca

-- cours: INF2110-10

------------------------------------------------------------------------------

PACKAGE lecture is

------------------------------------------------------------------------------

-- Lis un caractere et fait la validation du caractere a partir d'une chaine

-- de caractere.

-- Parametre: in: ListeCaractere: Liste des caractere valide.

-- Parametre: in: Question: Question qui est pose a l'usager.

-- retourne: character: Le caractere lus.

------------------------------------------------------------------------------

function LireCaractere(ListeCaracteres: in String; Question: in string) return character;

------------------------------------------------------------------------------

-- Lis une chaine avec un filtre de caracteres.

-- PARAMETRE: Filtre : (IN) Filtre des caracteres de la chaines. Doit etre la

-- meme longueur que la chaine retourne.

-- ' ' => N'importe quelle caractere.

-- 'N' => Nombre

-- 'L' => Lettre Majuscule

-- 'l' => Lettre miniscule

-- 'A' => Lettre sans tenir compte majuscule/minuscule

-- exemple: "LLLN N", le filtre vas obliger de saisir

-- 3 lettres majuscules

-- 1 nombre

-- 1 caractere quelconque

-- 1 nombre

-- Question: (IN) Question qui est pose a l'usager.

-- RETOURNE: La chaine lus.

------------------------------------------------------------------------------

FUNCTION LireChaine(Filtre: in String;Question: in String) return String;

------------------------------------------------------------------------------

-- Lis un nombre positif sur la ligne de commande.

-- Parametre: Question (IN): Question qui est pose a l'usager.

-- retourne: positive: nombre lus sur la ligne de commande.

------------------------------------------------------------------------------

function LirePositive(Question: in String) return Positive;

------------------------------------------------------------------------------

-- Cette procedure attends apres une touche

------------------------------------------------------------------------------

PROCEDURE AttendreTouche;

end lecture;

### LECTURE.ADB

------------------------------------------------------------------------------

--$Id: lecture.adb,v 1.1 2003/07/16 01:40:45 yann Exp $

------------------------------------------------------------------------------

-- Module qui fait une gestion evolue des entrees au clavier.

--

-- Auteur: Yann Bourdeau

-- Courriel: bourdeau.yann@courrier.uqam.ca

-- cours: INF2110-10

------------------------------------------------------------------------------

------------------------------------------------------------------------------

-- Liste des modules utilisees.

------------------------------------------------------------------------------

WITH ada.text\_io; use ada.text\_io;

PACKAGE body lecture is

package ES\_D\_Positive is new Integer\_IO(positive);

------------------------------------------------------------------------------

-- Lis un caractere et fait la validation du caractere a partir d'une chaine de caractere.

-- PARAMETRE: ListeCaractere: (IN) Liste des caractere valide.

-- Question: (IN) Question qui est pose a l'usager.

-- RETOURNE: character: Le caractere lus.

------------------------------------------------------------------------------

function LireCaractere(ListeCaracteres: in String;

Question: in string)

return character is

-- declaration des variables

Caractere: Character; -- Caractere lus a partir du clavier.

begin

principal: loop

begin

put(Question);

Get(Caractere);

skip\_line;

new\_line;

for i in ListeCaracteres'range loop

if(ListeCaracteres(i) = Caractere) then

exit Principal;

end if;

end loop;

put\_line("Caractere invalide");

new\_line;

exception

when Data\_Error =>

Put\_line("erreur de saisie de donnees.");

new\_line;

skip\_line;

end;

end loop principal;

return caractere;

end LireCaractere;

------------------------------------------------------------------------------

-- Lis une chaine avec un filtre de caracteres.

-- PARAMETRE: Filtre : (IN) Filtre des caracteres de la chaines. Doit etre la

-- meme longueur que la chaine retourne.

-- ' ' => N'importe quelle caractere.

-- 'N' => Nombre

-- 'L' => Lettre Majuscule

-- 'l' => Lettre miniscule

-- 'A' => Lettre sans tenir compte majuscule/minuscule

-- exemple: "LLLN N", le filtre vas obliger de saisir

-- 3 lettres majuscules

-- 1 nombre

-- 1 caractere quelconque

-- 1 nombre

-- Question: (IN) Question qui est pose a l'usager.

-- RETOURNE: La chaine lus.

------------------------------------------------------------------------------

FUNCTION LireChaine(Filtre: in String;

Question: in String)

return String is

-- Declaration des variables

TempChaine: string(1..Filtre'Last + 1); -- Chaine qui contient la chaine lus au clavier

Chaine: String(1..Filtre'Last);-- Chaine qui vas contenir la chaine retourner

LongueurChaine : Natural; -- Longueur de la chaine lus au clavier

ChaineValide : boolean; -- Determine si la chaine est valide ou pas

begin

loop

ChaineValide:=true;

put(Question);

Get\_Line(TempChaine, LongueurChaine);

-- remplis le reste de la chaine avec des espace blanc

For i in LongueurChaine+1..TempChaine'Last loop

TempChaine(i) := ' ';

End loop;

-- Si la chaine lus est au maximum, trop de caracteres

if (LongueurChaine>Chaine'Last) then

put\_line("Trop de caractere(s)");

ChaineValide := false;

-- Enleve les caracteres de trop dans le tampon

-- de lecture

skip\_line;

else

-- Valide chaque caractere de la chaine

FOR i in Filtre'Range loop

CASE Filtre(i) IS

WHEN ' ' =>

null;

WHEN 'N' =>

IF not (TempChaine(i) in '0'..'9') THEN

put\_line("Caractere invalide a la position" & Integer'Image(i) & ".");

ChaineValide := False;

END IF;

WHEN 'L' =>

IF not (TempChaine(i) in 'A'..'Z') THEN

put\_line("Caractere invalide a la position" & Integer'Image(i) & ".");

ChaineValide := false;

END IF;

WHEN 'l' =>

IF not (TempChaine(i) in 'a'..'z') THEN

put\_line("Caractere invalide a la position" & Integer'Image(i) & ".");

ChaineValide := false;

END IF;

WHEN 'A' =>

IF not (TempChaine(i) in 'a'..'z' or TempChaine(i) in 'A'..'Z') THEN

put\_line("Caractere invalide a la position" & Integer'Image(i) & ".");

ChaineValide := false;

END IF;

WHEN others =>

null;

END CASE;

end loop;

end if;

EXIT when ChaineValide;

end loop;

-- copie la chaine lus dans la destination.

Chaine := TempChaine(Chaine'First..Chaine'Last);

RETURN Chaine;

end LireChaine;

------------------------------------------------------------------------------

-- Lis un nombre positif sur la ligne de commande.

-- Parametre: Question (IN): Question qui est pose a l'usager.

-- retourne: positive: nombre lus sur la ligne de commande.

------------------------------------------------------------------------------

function LirePositive(Question: in String) return Positive is

Nombre: positive ; -- Nombre qui est lus sur le clavier.

begin

loop

begin

put(Question);

ES\_D\_Positive.get(Nombre);

skip\_line;

EXIT ;

exception

when Data\_error =>

Put\_line("Erreur de saisie de donnees. Le nombre doit etre un entier positif.");

new\_line;

skip\_line;

end;

end loop;

return Nombre;

end LirePositive;

------------------------------------------------------------------------------

-- Cette procedure attends apres une touche

------------------------------------------------------------------------------

PROCEDURE AttendreTouche is

begin

put\_line("Appuyez sur 'ENTER' pour continuer!");

skip\_line;

end AttendreTouche;

end lecture;

### LISTE\_DYN.ADS

-------------------------------------------------------------------------------

-- $Id: Liste\_Dyn.ads,v 1.2 2003/07/16 14:30:25 yann Exp $

-------------------------------------------------------------------------------

-- Sujet : Module de gestion de listes

-- Partie implantation à écrire

-- Date : Juillet 2003

-- Cours : inf2110 - travail pratique no.1

-- Note : le module ne retourne aucune exception au programme client

-- Le paramètre Succes retournera à faux dans le cas de l'insucces

-- du traitement du service.

-------------------------------------------------------------------------------

GENERIC

TYPE Type\_Element IS Private; -- Type de l'enregistrement a gerer

TYPE Type\_Cle IS Private; -- Type de la cle de l'enregistrement

WITH Function LaCle (Item : Type\_Element) Return Type\_Cle; -- Fonction qui retourne la valeur de la cle

WITH Function "<" (Gauche,Droite : Type\_Cle) Return Boolean; -- Importation de l'operateur < pour comparer des cles

PACKAGE Liste\_Dyn is

Type Liste is private; -- Type prive

Procedure Creer\_Liste (L : In Out Liste);

-------------------------------------------------------------------

-- Ant : la liste L peut deja contenir des elements a detruire

-- Cons : la liste L est à l'etat vide

-------------------------------------------------------------------

Function Liste\_Vide (L : in Liste) Return Boolean;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Retourne la valeur TRUE si la liste est vide ou False

-- si la liste contient au moins un element

-------------------------------------------------------------------

Procedure Detruire\_liste (L : in out Liste);

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Tous les elements de la liste sont detruits et l'etat

-- de la liste est indefini

-------------------------------------------------------------------

Function Nombre\_Elements (L : in Liste) Return Natural;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : retourne le nombre d'éléments de la liste

-------------------------------------------------------------------

--

-- Services de gestion d'élements

--

Procedure Inserer\_en\_Ordre (L : in out Liste; -- Liste

Element : In Type\_Element; -- l'element à inserer

Succes : out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : l'element est inséré dans la liste selon la valeur de la

-- cle de l'element. Succes retourne False, si un probleme

-- est survenu lors de l'insertion. Il ne peut y avoir 2

-- elements ayant la même valeur de clé.

-------------------------------------------------------------------

Procedure Modifier (L : in out Liste; -- Liste

Element : in Type\_Element; -- l'element à modifier

Succes : Out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Le contenu de l'élément déjà présent dans la liste

-- a été remplacée par l'élement en paramètre. Si l'élément

-- n'est pas dans la liste, succes retourne False.

-- Note : dans cette version, la valeur de la clé n'est pas modifiable

-------------------------------------------------------------------

Procedure Obtenir (L : in Liste; -- Liste

Cle : in Type\_cle; -- Cle de l'élement

Element : out Type\_Element; -- Element retourne

Succes : out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : on retourne l'élement correspondant à la clé fournie.

-- Succes retourne False, si aucun element n'est trouvé

-------------------------------------------------------------------

Procedure Retirer (L : in out Liste; -- Liste

Cle : in Type\_Cle; -- Cle de l'élément à retirer

Succes : out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : l'element correspondant à la clé est retiré de la liste

-- Succes retourne False si l'élement n'est pas trouve

-------------------------------------------------------------------

Function Premier (L : in Liste) Return Type\_Cle;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie et non vide

-- Cons : La clé du premier élément de la liste est retournée

-------------------------------------------------------------------

Function Dernier (L : in Liste) Return Type\_Cle;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie et non vide

-- Cons : La clé du dernier élément de la liste est retournée

-------------------------------------------------------------------

Procedure Successeur (L : in Liste; -- Liste

Cle\_Entree : in Type\_Cle; -- Clé en entrée

Cle\_Suivante : Out Type\_Cle; -- Clé de l'élement suivant

Succes : Out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Si l'élement correspondant à la cle\_entree existe,

-- la cle de l'élement suivant sera retournee.

-- Succes retourne False si l'élement n'a pas de successeur

-------------------------------------------------------------------

Procedure Predecesseur (L : in Liste; -- Liste

Cle\_Entree : in Type\_Cle; -- Clé en entrée

Cle\_Precedente: Out Type\_Cle; -- Clé de l'élement precedent

Succes : Out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Si l'élement correspondant à la cle\_entree existe,

-- la cle de l'élement precedent sera retournee.

-- Succes retourne False si l'élement n'a pas de predecesseur

-------------------------------------------------------------------

Procedure Sauvegarder (L : In Liste; -- Liste à sauvegarder

Nom : In String; -- Nom du fichier

Succes : Out Boolean); -- Succes de l'operation

---------------------------------------------------------------------

-- Sauvegarder les élements de la liste sur un fichier binaire

-- à accès séquentiel

--

-- Cons : Les élements de la liste sont sauvegardés

-- Succes retourne à TRUE si le traitement s'est bien déroulé

-- Succes retourne FALSE si une erreur est survenue

---------------------------------------------------------------------

Procedure Recuperer(L : Out Liste; -- Liste à récuperer

Nom : In String; -- Nom du fichier

Succes : Out Boolean); -- Succes de l'operation

---------------------------------------------------------------------

-- Recuperer les élements de la liste d'un fichier binaire

-- à accès séquentiel

--

-- Cons : Les élements de la liste sont recupérés dans la liste.

-- La liste est créée et son contenu préalable est perdu

-- Succes retourne à TRUE si le traitement s'est bien déroulé

-- Succes retourne FALSE si une erreur est survenue

---------------------------------------------------------------------

PRIVATE

Type Noeud;

Type PtrNoeud is ACCESS Noeud;

Type Liste is record

Tete : PtrNoeud;

Remorque : PtrNoeud; -- Pointe sur le noeud bidon de fin de liste

Nb\_elements : Natural := 0;

end record;

END Liste\_Dyn;

### LISTE\_DYN.ADB

------------------------------------------------------------------------------

--$Id: Liste\_Dyn.adb,v 1.2 2003/07/16 17:30:13 yann Exp $

------------------------------------------------------------------------------

-- Module qui fait la gestion de liste dynamique.

--

-- Auteur: Yann Bourdeau

-- Courriel: bourdeau.yann@courrier.uqam.ca

-- cours: INF2110-10

------------------------------------------------------------------------------

------------------------------------------------------------------------------

-- Liste des modules utilisees.

------------------------------------------------------------------------------

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Sequential\_IO;

WITH Unchecked\_deallocation;

PACKAGE body Liste\_Dyn is

Type Noeud is record

Suivant: PtrNoeud;

Element: Type\_Element;

end record;

PACKAGE ES\_Type\_Element\_Bin\_Seq is new ada.sequential\_IO(Type\_Element);

PROCEDURE Liberer\_Noeud IS New Unchecked\_deallocation (Noeud, PtrNoeud);

------------------------------------------------------------------------------

-- Cette procedure Libere une liste chaine.

-- NIVEAU: 2

-- PARAMETRE: (IN) L: Liste a etre liberer.

------------------------------------------------------------------------------

PROCEDURE LibereMemoireListe(Courant: in out PtrNoeud) is

Suivant: PtrNoeud; -- Pointeur sur le noeud suivant.

begin

WHILE Courant /= NULL LOOP

Suivant := Courant.Suivant;

Liberer\_Noeud(Courant);

Courant:=Suivant;

END LOOP;

end;

------------------------------------------------------------------------------

-- Cette procedure initilise une liste.

-- NIVEAU: 1

-- PARAMETRE: (IN) L: Liste a etre initialise.

------------------------------------------------------------------------------

Procedure Creer\_Liste (L : In Out Liste) is

begin

IF L.Tete /= NULL THEN

LibereMemoireListe(L.Tete);

L.Remorque:=null;

L.Nb\_elements := 0;

END IF;

l.Tete := new Noeud;

l.Remorque:= l.Tete;

l.Tete.Suivant := null;

l.Nb\_elements := 0;

end Creer\_Liste;

------------------------------------------------------------------------------

-- Cette procedure determine si une liste est vide.

-- NIVEAU: 1

-- PARAMETRE: (IN) L: Liste a etre verifier

-- RETOURNE: TRUE => La liste est vide.

-- FALSE => La liste est non vide.

------------------------------------------------------------------------------

Function Liste\_Vide (L : in Liste) Return Boolean is

BEGIN

IF l.Nb\_elements < 1 THEN

RETURN true;

ELSE

RETURN false;

END IF;

END Liste\_Vide;

------------------------------------------------------------------------------

-- Cette procedure detruis la liste.

-- NIVEAU: 1

-- Parametre: (IN) L: Liste a detruire.

------------------------------------------------------------------------------

Procedure Detruire\_liste (L : in out Liste) is

Begin

IF L.Tete /= NULL THEN

LibereMemoireListe(L.Tete);

L.Remorque:=null;

L.Nb\_elements := 0;

END IF;

End Detruire\_liste;

------------------------------------------------------------------------------

-- Cette procedure retourne le nombre d'element contenus dans la liste.

-- NIVEAU: 1

-- Parametre: (IN) L: Liste a retourne le nombre d'element.

------------------------------------------------------------------------------

Function Nombre\_Elements (L : in Liste) Return Natural is

begin

RETURN l.Nb\_elements;

end Nombre\_Elements;

------------------------------------------------------------------------------

-- Cette procedure insere un element en ordre dans la liste.

-- NIVEAU: 1

-- Parametre: (IN OUT)L: Liste qui vas contenir le nouveau element.

-- (IN) Element: Element a ajouter.

-- (OUT) succes: TRUE => Operation a ete un succes.

-- FALSE => erreur.

------------------------------------------------------------------------------

Procedure Inserer\_en\_Ordre (L : in out Liste;

Element : In Type\_Element;

Succes : out Boolean) is

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

trouve: boolean := false; -- Determine si une element plus grand a ete trouve.

pNoeud: PtrNoeud; -- Pointeur sur le nouveau noeud.

begin

succes := true;

pCourant := l.Tete;

-- Cherche le premiere position plus grande.

WHILE (pCourant /= l.Remorque) and (not trouve) LOOP

IF LaCle(pCourant.Element) < LaCle(Element) THEN

pCourant:= pCourant.Suivant;

ELSE

trouve := true;

END IF;

END LOOP;

-- Allocation du nouveau noeud.

pNoeud := new Noeud;

pNoeud.Suivant := null;

IF not TROUVE THEN

-- Si aucune element trouve plus grand, insere a la fin.

l.Remorque.Element := Element;

l.Remorque.Suivant := pnoeud;

l.Remorque := pnoeud;

ELSE

-- Sinon insere au debut.

pNoeud.Element := pCourant.Element;

pNoeud.Suivant := pCourant.Suivant;

pCourant.Element := element;

pCourant.Suivant := pNoeud;

END IF;

l.Nb\_elements := l.Nb\_elements + 1;

end Inserer\_en\_Ordre;

------------------------------------------------------------------------------

-- Cette procedure permet de modifier un enregistrement existant.

-- NIVEAU: 1

-- PARAMETRE: L: (IN OUT) Liste qui contient l'enregistrement a etre modifie.

-- Element: (IN) Element a etre modifie.

-- Succes: (OUT) VRAI => L'element a ete modifie.

-- FAUX => L'element n'as pas ete modifie.

------------------------------------------------------------------------------

Procedure Modifier (L : in out Liste;

Element : in Type\_Element;

Succes : Out Boolean) is

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

trouve : boolean:= false; -- Determine si l'enregistrement a ete trouve.

begin

Succes := false;

pCourant := l.Tete;

-- Cherche l'element.

WHILE (pCourant /= l.Remorque) and (not trouve) LOOP

IF LaCle(pCourant.Element) = LaCle(Element) THEN

trouve := true;

ELSE

pCourant := pCourant.Suivant;

END IF;

END LOOP;

-- Si l'element a ete trouve, modifie le.

IF trouve THEN

pCourant.Element := Element;

succes := true;

END IF;

end Modifier;

------------------------------------------------------------------------------

-- Cette procedure retourne un element de la liste.

-- NIVEAU: 1

-- PARAMETRE: L: (IN) Liste qui contient l'enregistrement a etre retourne.

-- Cle: (IN) Cle de l'enregistrement a etre retourne.

-- Element: (OUT) Contient l'element retourne.

-- Succes: (OUT) VRAI => Si l'element a etre trouve.

-- FAUX => L'element n'as pas ete trouve.

------------------------------------------------------------------------------

Procedure Obtenir (L : in Liste;

Cle : in Type\_cle;

Element : out Type\_Element;

Succes : out Boolean) is

pCourant: PtrNoeud;-- Pointeur sur le noeud courant.

begin

pCourant := l.Tete;

succes := false;

-- Checher l'element.

WHILE (pCourant /= l.Remorque) and ( not succes) LOOP

IF LaCle(pCourant.Element) = Cle THEN

-- Trouve!

Element := pCourant.Element;

succes := true;

ELSE

pCourant := pCourant.Suivant;

END IF;

END LOOP;

end Obtenir;

------------------------------------------------------------------------------

-- Cette procedure permet de retirer un element de la liste.

-- NIVEAU: 1

-- PARAMETRE: L: (IN OUT) Liste qui contient l'enregistrement a etre enlever.

-- Cle: (IN) Cle de l'enregistrement a etre enlever.

-- Succes: (OUT) VRAI => Si l'enregistrement a ete enlever.

-- FAUX => Si l'enregistrement n'as pas ete enlever.

------------------------------------------------------------------------------

Procedure Retirer (L : in out Liste;

Cle : in Type\_Cle;

Succes : out Boolean) is

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

pSuivant: PtrNoeud; -- Pointeur sur le noeud suivant le courant.

begin

succes := false;

pCourant := l.Tete;

-- Cherche l'element a retirer!

WHILE pCourant /= l.Remorque and not succes LOOP

IF LaCle(pCourant.Element) = Cle THEN

-- Trouver

Succes := true;

pSuivant := pCourant.Suivant;

pCourant.all := pSuivant.all;

IF pSuivant = l.Remorque THEN

l.Remorque := pCourant;

END IF;

Liberer\_Noeud(pSuivant);

END IF;

END LOOP;

end Retirer;

------------------------------------------------------------------------------

-- Cette fonction retourne la cle du premier element.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient l'enregistrement a etre retourner.

-- RETOURN: CLE: La cle du premier element de la liste.

------------------------------------------------------------------------------

Function Premier (L : in Liste) Return Type\_Cle is

begin

return LaCle (l.Tete.Element);

end Premier;

------------------------------------------------------------------------------

-- Cette fonction retourne la cle du dernier element de la liste.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient l'enregistrement a etre retourner.

-- RETOURN: CLE: La cle du dernier element de la liste.

------------------------------------------------------------------------------

Function Dernier (L : in Liste) Return Type\_Cle is

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

begin

pCourant := l.Tete;

-- Vas au dernier element.

WHILE pCourant.suivant /= l.Remorque LOOP

pCourant := pCourant.Suivant;

END LOOP;

RETURN LaCle(pCourant.Element);

end Dernier;

------------------------------------------------------------------------------

-- Cette procedure retourne la cle de l'enregistrement suivant celui specifie

-- par la cle.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient les enregistrements.

-- Cle\_Entree: (IN) Cle de l'enregistrement a trouver.

-- Cle\_Suivante: (OUT) Cle de l'enregistrement qui suit celui

-- qui a ete trouve.

-- Succes: (OUT) VRAI => L'enregistrement a ete trouve.

-- FAUX => L'enregistrement n'as pas ete trouve.

------------------------------------------------------------------------------

Procedure Successeur (L : in Liste;

Cle\_Entree : in Type\_Cle;

Cle\_Suivante : Out Type\_Cle;

Succes : Out Boolean) is

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

begin

succes := false;

pCourant := l.Tete;

-- Vas a l'element suivant la cle specifie.

WHILE (pCourant /= l.Remorque) and (not succes ) LOOP

IF LaCle(pCourant.Element) = Cle\_entree THEN

-- Trouve

-- Si l'element suivant n'est pas la remorque!

IF pCourant.Suivant /= l.Remorque THEN

-- OK!

succes := true;

Cle\_suivante := LaCle(pCourant.Suivant.Element);

END IF;

END IF;

pCourant := pCourant.Suivant;

END LOOP;

end Successeur;

------------------------------------------------------------------------------

-- Cette procedure retourne la cle de l'enregistrement qui precede celui

-- specifie.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient les enregistrement.

-- Cle\_Entree:(IN) Cle de l'enregistrement a trouver.

-- Cle\_Precedente:(OUT) Cle de l'enregistrement qui precede celui

-- qui a ete specifie.

-- Succes: (OUT) VRAI => L'enregistrement a ete trouver.

-- FAIX => L'enregistrement n'as pas ete trouver.

------------------------------------------------------------------------------

Procedure Predecesseur (L : in Liste;

Cle\_Entree : in Type\_Cle;

Cle\_Precedente: Out Type\_Cle;

Succes : Out Boolean) is

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

begin

succes := false;

-- Si non liste vide.

if (l.Tete /= l.Remorque) then

pCourant := l.Tete;

-- Vas a l'element chercher.

WHILE (pCourant.Suivant /= l.Remorque) and (not succes ) LOOP

-- Verifie si element = cle.

IF LaCle(pCourant.Suivant.Element) = Cle\_entree THEN

succes := true;

Cle\_Precedente := LaCle(pCourant.Element);

END IF;

pCourant := pCourant.Suivant;

END LOOP;

end if;

end Predecesseur;

------------------------------------------------------------------------------

-- Procedure qui sauvegarde la liste dans un fichier.

-- NIVEAU: 1

-- Parametre: (IN) Liste: liste a sauvegarder.

-- (IN) Nom: Nom du fichier qui vas contenir la liste

-- (OUT) Succes: VRAI => Le fichier a ete sauvegarder.

-- FAUX => Le fichier n'as pas ete sauvegarder.

------------------------------------------------------------------------------

Procedure Sauvegarder (L : In Liste;

Nom : In String;

Succes : Out Boolean) is

-- Declaration des variables.

Fichier: ES\_Type\_Element\_Bin\_Seq.File\_Type; -- Definition du fichier

pCourant: PtrNoeud; -- Pointeur sur le noeud courant.

begin

succes:=true;

ES\_Type\_Element\_Bin\_Seq.Create(Fichier,ES\_Type\_Element\_Bin\_Seq.Out\_File,Nom);

pCourant := l.Tete;

WHILE pCourant /= l.Remorque LOOP

ES\_Type\_Element\_Bin\_Seq.Write(Fichier, pCourant.Element);

pCourant := pCourant.Suivant;

END LOOP;

ES\_Type\_Element\_Bin\_Seq.Close(Fichier);

exception

WHEN others => succes := false;

end Sauvegarder;

------------------------------------------------------------------------------

-- Procedure qui recupere la liste dans un fichier.

-- NIVEAU: 1

-- PARAMETRE: Liste: (IN) Liste qui vas contenir les elements lus.

-- Nom: (IN) Nom du fichier qui contient la liste.

-- (OUT) Succes: VRAI => Le fichier a ete lus.

-- FAUX => Le fichier n'as pas ete lus.

------------------------------------------------------------------------------

Procedure Recuperer(L : Out Liste;

Nom : In String;

Succes : Out Boolean) is

-- Declaration des variables.

Fichier: ES\_Type\_Element\_Bin\_Seq.File\_Type; -- Definition du fichier

pNoeud: PtrNoeud; -- Pointeur sur le noeud a ajouter.

begin

succes := true;

IF L.Tete /= NULL THEN

LibereMemoireListe(L.Tete);

L.Remorque:=null;

L.Nb\_elements := 0;

END IF;

pNoeud := new Noeud;

pNoeud.Suivant := NULL;

L.Tete := pNoeud;

ES\_Type\_Element\_Bin\_Seq.Open (Fichier, ES\_Type\_Element\_Bin\_Seq.In\_File,Nom);

while not ES\_Type\_Element\_Bin\_Seq.End\_Of\_File(Fichier) loop

L.Nb\_elements:=L.Nb\_elements+1;

ES\_Type\_Element\_Bin\_Seq.Read(Fichier,pNoeud.Element);

pNoeud.Suivant := new noeud;

pNoeud := pNoeud.Suivant;

end loop;

ES\_Type\_Element\_Bin\_Seq.Close(Fichier);

l.Remorque := pNoeud;

exception

WHEN others => succes := false;

l.Remorque := pNoeud; -- Etre sur que la liste soit initialise.

end Recuperer;

end Liste\_Dyn;

### LISTES\_TAB.ADS

-------------------------------------------------------------------------------

-- $Id: Listes\_tab.ads,v 1.1 2003/07/16 01:40:45 yann Exp $

-------------------------------------------------------------------------------

-- Sujet : Module de gestion de listes

-- Partie implantation à écrire

-- Date : Mai 2003

-- Cours : inf2110 - travail pratique no.1

-- Note : le module ne retourne aucune exception au programme client

-- Le paramètre Succes retournera à faux dans le cas de l'insucces

-- du traitement du service.

-------------------------------------------------------------------------------

GENERIC

Max\_Elements : Natural := 100; -- Nombre d'élements max. dans la liste

TYPE Type\_Element IS Private; -- Type de l'enregistrement a gerer

TYPE Type\_Cle IS Private; -- Type de la cle de l'enregistrement

WITH Function LaCle (Item : Type\_Element) Return Type\_Cle; -- Fonction qui retourne la valeur de la cle

WITH Function "<" (Gauche,Droite : Type\_Cle) Return Boolean; -- Importation de l'operateur < pour comparer des cles

PACKAGE Listes\_Tab is

Type Liste is private; -- Type prive

Procedure Creer\_Liste (L : In Out Liste);

-------------------------------------------------------------------

-- Ant : la liste L peut deja contenir des elements a detruire

-- Cons : la liste L est à l'etat vide

-------------------------------------------------------------------

Function Liste\_Vide (L : in Liste) Return Boolean;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Retourne la valeur TRUE si la liste est vide ou False

-- si la liste contient au moins un element

-------------------------------------------------------------------

Procedure Detruire\_liste (L : in out Liste);

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Tous les elements de la liste sont detruits et l'etat

-- de la liste est indefini

-------------------------------------------------------------------

Function Nombre\_Elements (L : in Liste) Return Natural;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : retourne le nombre d'éléments de la liste

-------------------------------------------------------------------

--

-- Services de gestion d'élements

--

Procedure Inserer\_en\_Ordre (L : in out Liste; -- Liste

Element : In Type\_Element; -- l'element à inserer

Succes : out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : l'element est inséré dans la liste selon la valeur de la

-- cle de l'element. Succes retourne False, si un probleme

-- est survenu lors de l'insertion. Il ne peut y avoir 2

-- elements ayant la même valeur de clé.

-------------------------------------------------------------------

Procedure Modifier (L : in out Liste; -- Liste

Element : in Type\_Element; -- l'element à modifier

Succes : Out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Le contenu de l'élément déjà présent dans la liste

-- a été remplacée par l'élement en paramètre. Si l'élément

-- n'est pas dans la liste, succes retourne False.

-- Note : dans cette version, la valeur de la clé n'est pas modifiable

-------------------------------------------------------------------

Procedure Obtenir (L : in Liste; -- Liste

Cle : in Type\_cle; -- Cle de l'élement

Element : out Type\_Element; -- Element retourne

Succes : out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : on retourne l'élement correspondant à la clé fournie.

-- Succes retourne False, si aucun element n'est trouvé

-------------------------------------------------------------------

Procedure Retirer (L : in out Liste; -- Liste

Cle : in Type\_Cle; -- Cle de l'élément à retirer

Succes : out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : l'element correspondant à la clé est retiré de la liste

-- Succes retourne False si l'élement n'est pas trouve

-------------------------------------------------------------------

Function Premier (L : in Liste) Return Type\_Cle;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie et non vide

-- Cons : La clé du premier élément de la liste est retournée

-------------------------------------------------------------------

Function Dernier (L : in Liste) Return Type\_Cle;

-------------------------------------------------------------------

-- Ant : la liste L doit être definie et non vide

-- Cons : La clé du dernier élément de la liste est retournée

-------------------------------------------------------------------

Procedure Successeur (L : in Liste; -- Liste

Cle\_Entree : in Type\_Cle; -- Clé en entrée

Cle\_Suivante : Out Type\_Cle; -- Clé de l'élement suivant

Succes : Out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Si l'élement correspondant à la cle\_entree existe,

-- la cle de l'élement suivant sera retournee.

-- Succes retourne False si l'élement n'a pas de successeur

-------------------------------------------------------------------

Procedure Predecesseur (L : in Liste; -- Liste

Cle\_Entree : in Type\_Cle; -- Clé en entrée

Cle\_Precedente: Out Type\_Cle; -- Clé de l'élement precedent

Succes : Out Boolean); -- Succes de l'operation

-------------------------------------------------------------------

-- Ant : la liste L doit être definie

-- Cons : Si l'élement correspondant à la cle\_entree existe,

-- la cle de l'élement precedent sera retournee.

-- Succes retourne False si l'élement n'a pas de predecesseur

-------------------------------------------------------------------

Procedure Sauvegarder (L : In Liste; -- Liste à sauvegarder

Nom : In String; -- Nom du fichier

Succes : Out Boolean); -- Succes de l'operation

---------------------------------------------------------------------

-- Sauvegarder les élements de la liste sur un fichier binaire

-- à accès séquentiel

--

-- Cons : Les élements de la liste sont sauvegardés

-- Succes retourne à TRUE si le traitement s'est bien déroulé

-- Succes retourne FALSE si une erreur est survenue

---------------------------------------------------------------------

Procedure Recuperer(L : Out Liste; -- Liste à récuperer

Nom : In String; -- Nom du fichier

Succes : Out Boolean); -- Succes de l'operation

---------------------------------------------------------------------

-- Recuperer les élements de la liste d'un fichier binaire

-- à accès séquentiel

--

-- Cons : Les élements de la liste sont recupérés dans la liste.

-- La liste est créée et son contenu préalable est perdu

-- Succes retourne à TRUE si le traitement s'est bien déroulé

-- Succes retourne FALSE si une erreur est survenue

---------------------------------------------------------------------

PRIVATE

Type Type\_tableau is Array (1..Max\_elements) Of Type\_Element;

Type Liste is record

Tab : Type\_Tableau;

Nb\_elements : Natural := 0;

end record;

END Listes\_Tab;

### LISTES\_TAB.ADB

------------------------------------------------------------------------------

--$Id: Listes\_tab.adb,v 1.1 2003/07/16 01:40:45 yann Exp $

------------------------------------------------------------------------------

-- Module qui fait la gestion de liste.

--

-- Auteur: Yann Bourdeau

-- Courriel: bourdeau.yann@courrier.uqam.ca

-- cours: INF2110-10

------------------------------------------------------------------------------

------------------------------------------------------------------------------

-- Liste des modules utilisees.

------------------------------------------------------------------------------

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Sequential\_IO;

PACKAGE body Listes\_Tab is

PACKAGE ES\_Type\_Element\_Bin\_Seq is new ada.sequential\_IO(Type\_Element);

------------------------------------------------------------------------------

-- Cette procedure initilise une liste.

-- NIVEAU: 1

-- PARAMETRE: (IN) L: Liste a etre initialise.

------------------------------------------------------------------------------

Procedure Creer\_Liste (L : In Out Liste) is

begin

L.Nb\_elements:=0;

end Creer\_Liste;

------------------------------------------------------------------------------

-- Cette procedure determine si une liste est vide.

-- NIVEAU: 1

-- PARAMETRE: (IN) L: Liste a etre verifier

-- RETOURNE: TRUE => La liste est vide.

-- FALSE => La liste est non vide.

------------------------------------------------------------------------------

Function Liste\_Vide (L : in Liste) Return Boolean is

BEGIN

if(L.Nb\_elements = 0) then

RETURN TRUE;

else

RETURN FALSE;

end if;

END Liste\_Vide;

------------------------------------------------------------------------------

-- Cette procedure detruis la liste.

-- NIVEAU: 1

-- Parametre: (IN) L: Liste a detruire.

------------------------------------------------------------------------------

Procedure Detruire\_liste (L : in out Liste) is

Begin

L.Nb\_elements :=0;

End Detruire\_liste;

------------------------------------------------------------------------------

-- Cette procedure retourne le nombre d'element contenus dans la liste.

-- NIVEAU: 1

-- Parametre: (IN) L: Liste a retourne le nombre d'element.

------------------------------------------------------------------------------

Function Nombre\_Elements (L : in Liste) Return Natural is

begin

RETURN L.Nb\_elements;

end Nombre\_Elements;

------------------------------------------------------------------------------

-- Cette procedure insere un element en ordre dans la liste.

-- NIVEAU: 1

-- Parametre: (IN OUT)L: Liste qui vas contenir le nouveau element.

-- (IN) Element: Element a ajouter.

-- (OUT) succes: TRUE => Operation a ete un succes.

-- FALSE => erreur.

------------------------------------------------------------------------------

Procedure Inserer\_en\_Ordre (L : in out Liste;

Element : In Type\_Element;

Succes : out Boolean) is

-- Declaration des variables.

ExisteDeja : boolean := false;

begin

-- Si le tableau est vide, inserer a la premiere position.

if(l.Nb\_elements = 0) then

L.tab(1):=Element;

l.Nb\_Elements := L.Nb\_Elements + 1;

succes := true;

else

Succes:= false;

FOR i in 1..L.Nb\_elements loop

-- Parcoure la liste jusqu'a ce que la cle de l'element soit

-- inferier a celle dans le tableau.

IF (not succes and not ExisteDeja) THEN

-- Verifie si elle n'existe deja pas.

if ( LaCle(L.Tab(i)) = LaCle(Element)) then

ExisteDeja := true;

end if;

-- Agrandie le tableau de 1.

IF (LaCle(Element)<(LaCle(L.tab(i) ) )) THEN

FOR z in reverse i..L.Nb\_elements loop

L.tab(z+1):=l.tab(z);

end loop;

L.tab(i):=Element;

L.Nb\_Elements:=L.Nb\_Elements+1;

succes := true;

END IF;

END IF;

end loop;

-- Verifie si l'element n'as pas ete inserer puisque le dernier

IF not succes and (LaCle(l.tab(l.Nb\_elements)) < LaCle(Element) )THEN

l.Nb\_elements:=l.Nb\_Elements+1;

l.Tab(l.Nb\_elements):=Element;

Succes:=true;

END IF;

end if;

exception

WHEN others => succes := false;

end Inserer\_en\_Ordre;

------------------------------------------------------------------------------

-- Cette procedure permet de modifier un enregistrement existant.

-- NIVEAU: 1

-- PARAMETRE: L: (IN OUT) Liste qui contient l'enregistrement a etre modifie.

-- Element: (IN) Element a etre modifie.

-- Succes: (OUT) VRAI => L'element a ete modifie.

-- FAUX => L'element n'as pas ete modifie.

------------------------------------------------------------------------------

Procedure Modifier (L : in out Liste;

Element : in Type\_Element;

Succes : Out Boolean) is

begin

Succes:= false;

FOR i in 1..L.Nb\_Elements loop

IF (Not succes) and (LaCle(l.Tab(i)) = LaCle(Element) ) THEN

succes := true;

l.tab(i):=Element;

END IF;

end loop;

exception

WHEN others => succes := false;

end Modifier;

------------------------------------------------------------------------------

-- Cette procedure retourne un element de la liste.

-- NIVEAU: 1

-- PARAMETRE: L: (IN) Liste qui contient l'enregistrement a etre retourne.

-- Cle: (IN) Cle de l'enregistrement a etre retourne.

-- Element: (OUT) Contient l'element retourne.

-- Succes: (OUT) VRAI => Si l'element a etre trouve.

-- FAUX => L'element n'as pas ete trouve.

------------------------------------------------------------------------------

Procedure Obtenir (L : in Liste;

Cle : in Type\_cle;

Element : out Type\_Element;

Succes : out Boolean) is

begin

succes:= false;

FOR i in 1..L.Nb\_Elements loop

IF (not succes) and (LaCle(l.tab(i)) = Cle ) THEN

Succes:= true;

Element:=l.tab(i);

END IF;

end loop;

exception

WHEN others => succes := false;

end Obtenir;

------------------------------------------------------------------------------

-- Cette procedure permet de retirer un element de la liste.

-- NIVEAU: 1

-- PARAMETRE: L: (IN OUT) Liste qui contient l'enregistrement a etre enlever.

-- Cle: (IN) Cle de l'enregistrement a etre enlever.

-- Succes: (OUT) VRAI => Si l'enregistrement a ete enlever.

-- FAUX => Si l'enregistrement n'as pas ete enlever.

------------------------------------------------------------------------------

Procedure Retirer (L : in out Liste;

Cle : in Type\_Cle;

Succes : out Boolean) is

begin

succes:=false;

FOR i in 1..L.Nb\_Elements loop

IF (not succes) and ( LaCle(l.Tab(i))=cle) THEN

FOR z in i..L.Nb\_Elements-1 loop

L.Tab(z):=l.Tab(z+1);

end loop;

succes:= true;

L.Nb\_Elements:=L.Nb\_Elements-1;

END IF;

end loop;

exception

WHEN others => succes := false;

end Retirer;

------------------------------------------------------------------------------

-- Cette fonction retourne la cle du premier element.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient l'enregistrement a etre retourner.

-- RETOURN: CLE: La cle du premier element de la liste.

------------------------------------------------------------------------------

Function Premier (L : in Liste) Return Type\_Cle is

begin

--IF (l.Nb\_Elements >0 ) THEN

RETURN LaCle(L.tab(1));

--END IF;

end Premier;

------------------------------------------------------------------------------

-- Cette fonction retourne la cle du dernier element de la liste.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient l'enregistrement a etre retourner.

-- RETOURN: CLE: La cle du dernier element de la liste.

------------------------------------------------------------------------------

Function Dernier (L : in Liste) Return Type\_Cle is

begin

--IF (l.Nb\_elements >0 ) THEN

RETURN LaCle(l.tab(l.Nb\_Elements));

--END IF;

end Dernier;

------------------------------------------------------------------------------

-- Cette procedure retourne la cle de l'enregistrement suivant celui specifie

-- par la cle.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient les enregistrements.

-- Cle\_Entree: (IN) Cle de l'enregistrement a trouver.

-- Cle\_Suivante: (OUT) Cle de l'enregistrement qui suit celui

-- qui a ete trouve.

-- Succes: (OUT) VRAI => L'enregistrement a ete trouve.

-- FAUX => L'enregistrement n'as pas ete trouve.

------------------------------------------------------------------------------

Procedure Successeur (L : in Liste;

Cle\_Entree : in Type\_Cle;

Cle\_Suivante : Out Type\_Cle;

Succes : Out Boolean) is

begin

succes := false;

FOR i in 1..L.Nb\_Elements loop

IF (NOT succes) and (LaCle(l.tab(i)) = Cle\_Entree) THEN

if ( i + 1 <= l.Nb\_Elements) then

succes:=true;

Cle\_Suivante := LaCle(L.Tab(i+1));

end if;

END IF;

end loop;

exception

WHEN others => succes := false;

end Successeur;

------------------------------------------------------------------------------

-- Cette procedure retourne la cle de l'enregistrement qui precede celui

-- specifie.

-- NIVEAU: 1

-- PARAMETRE: L : (IN) Liste qui contient les enregistrement.

-- Cle\_Entree:(IN) Cle de l'enregistrement a trouver.

-- Cle\_Precedente:(OUT) Cle de l'enregistrement qui precede celui

-- qui a ete specifie.

-- Succes: (OUT) VRAI => L'enregistrement a ete trouver.

-- FAIX => L'enregistrement n'as pas ete trouver.

------------------------------------------------------------------------------

Procedure Predecesseur (L : in Liste;

Cle\_Entree : in Type\_Cle;

Cle\_Precedente: Out Type\_Cle;

Succes : Out Boolean) is

begin

succes:= false;

FOR i in 1..L.Nb\_Elements loop

IF (Not succes) and (LaCle(L.Tab(i)) = Cle\_Entree) THEN

IF (i-1 > 0) THEN

succes:=true;

Cle\_Precedente := LaCle(l.Tab(i-1));

END IF;

END IF;

end loop;

exception

WHEN others => succes := false;

end Predecesseur;

------------------------------------------------------------------------------

-- Procedure qui sauvegarde la liste dans un fichier.

-- NIVEAU: 1

-- Parametre: (IN) Liste: liste a sauvegarder.

-- (IN) Nom: Nom du fichier qui vas contenir la liste

-- (OUT) Succes: VRAI => Le fichier a ete sauvegarder.

-- FAUX => Le fichier n'as pas ete sauvegarder.

------------------------------------------------------------------------------

Procedure Sauvegarder (L : In Liste;

Nom : In String;

Succes : Out Boolean) is

-- Declaration des variables.

Fichier: ES\_Type\_Element\_Bin\_Seq.File\_Type; -- Definition du fichier

begin

succes:=true;

ES\_Type\_Element\_Bin\_Seq.Create(Fichier,ES\_Type\_Element\_Bin\_Seq.Out\_File,Nom);

For i in 1..L.Nb\_elements loop

ES\_Type\_Element\_Bin\_Seq.Write(Fichier, l.Tab(I));

End loop;

ES\_Type\_Element\_Bin\_Seq.Close(Fichier);

exception

WHEN others => succes := false;

end Sauvegarder;

------------------------------------------------------------------------------

-- Procedure qui recupere la liste dans un fichier.

-- NIVEAU: 1

-- PARAMETRE: Liste: (IN) Liste qui vas contenir les elements lus.

-- Nom: (IN) Nom du fichier qui contient la liste.

-- (OUT) Succes: VRAI => Le fichier a ete lus.

-- FAUX => Le fichier n'as pas ete lus.

------------------------------------------------------------------------------

Procedure Recuperer(L : Out Liste;

Nom : In String;

Succes : Out Boolean) is

-- Declaration des variables.

Fichier: ES\_Type\_Element\_Bin\_Seq.File\_Type; -- Definition du fichier

begin

succes := true;

L.Nb\_elements:=0;

ES\_Type\_Element\_Bin\_Seq.Open (Fichier, ES\_Type\_Element\_Bin\_Seq.In\_File,Nom);

while not ES\_Type\_Element\_Bin\_Seq.End\_Of\_File(Fichier) loop

L.Nb\_elements:=L.Nb\_elements+1;

ES\_Type\_Element\_Bin\_Seq.Read(Fichier,l.Tab(L.Nb\_elements));

end loop;

ES\_Type\_Element\_Bin\_Seq.Close(Fichier);

exception

WHEN others => succes := false;

end Recuperer;

end Listes\_Tab;Annexe B – Script de test et fichiers de test

Test.cmd

echo off

rem $Id: test.cmd,v 1.2 2003/06/01 21:43:44 yannb Exp $

date /t

time /t

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 1: Création d'un cours et liste le cours.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn1.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 2: Création d'un nouveau dossier étudiant et sa consultation.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn2.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 3: Annulation d'un cours sans inscription.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn3.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 4: Ajout d'une inscription à un dossier étudiant

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn4.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 5: Annulation d'une inscription à un dossier étudiant

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn5.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 6: Annulation d'un cours avec inscription.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn6.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 7: Lister les inscriptions d'un cours

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn7.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas normaux

echo test 8: Sauvegarde et lecture des fichiers cours.bin et etudiant.bin

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testn81.txt

echo ===============================================

echo === Test8: Fichier sauvegarder et recharger ===

echo ===============================================

tp2 < testn82.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas limites

echo test 1: Création du maximum (100) de cours

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testl1.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas limites

echo test 2: Création du maximum (100) de dossiers etudiant.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testl2.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas limites

echo test 3: Création du maximum de dossier étudiant avec le même nom et prenom.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testl3.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas limites

echo test 4: Création du maximum de dossier étudiant avec le même nom et prenom.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testl4.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 1: Création de doublon de cours

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh1.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 2: Création de doublon d'inscription de cours pour un même étudiant

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh2.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 3: Annulation d'un cours qui n'existe pas.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh3.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 4: Annulation d'une inscription qui n'existe pas.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh4.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 5: Consultation d'un dossier qui n'existe pas

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh5.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 6: Ajouter une inscription a un cours qui n'existe pas.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh6.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 7: Liste des inscriptions d'un cours qui n'existe pas.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh7.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 8: Valider la saisie des champs de créer cours

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh8.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 9: Valider la saisie des champs d'annuler un cours.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh9.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 10: Valider la saisie des champs de liste des inscriptions.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh10.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 11: Valider la saisie des champs de créer un nouveau dossier

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh11.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 12: Valider la saisie des champs de consulter un dossier.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh12.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 13: Valider la saisie des champs d'ajouter une inscription.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh13.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 14: Valider la saisie des champs d'annuler une inscription.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh14.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 15: Valider la saisie des options du menu principal.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh15.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 16: Valider la saisie des options du menu cours

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh16.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

echo Verification des cas hors bornes.

echo test 17: Valider la saisie des options du menu étudiant.

copy cours.vide.bin cours.bin

copy etudiant.vide.bin etudiant.bin

tp2 < testh17.txt

echo ==========================================================================

echo ==========================================================================

echo ==========================================================================

date /t

time /t

### testn1.txt

c

c

inf1110

1

30

jeudi 15:00

1

l

q

q

### testn2.txt

e

c

Bourdeau

Yann

o

bou0

q

q

### testn3.txt

c

c

inf1110

1

30

jeudi 15:00

1

a

1

inf1110

o

l

q

q

### testn4.txt

e

c

Bourdeau

Yann

q

c

c

inf1110

1

30

jeudi 15:00

1

q

e

a

bou0

1

inf1110

o

o

bou0

q

c

i

1

inf1110

q

q

### testn5.txt

e

c

Bourdeau

Yann

q

c

c

inf1110

1

30

jeudi 15:00

1

q

e

a

bou0

1

inf1110

o

u

bou0

1

INF1110

o

o

bou0

q

q

### testn6.txt

e

c

Bourdeau

Yann

q

c

c

inf1110

1

30

jeudi 15:00

1

q

e

a

bou0

1

inf1110

o

q

c

a

1

inf1110

o

q

e

o

bou0

q

q

### testn7.txt

e

c

Bourdeau

Yann

q

c

c

inf1110

1

30

jeudi 15:00

1

q

e

a

bou0

1

inf1110

o

q

c

i

1

inf1110

q

q

### testn81.txt

e

c

Bourdeau

Yann

q

c

c

inf1110

1

30

jeudi 15:00

1

q

e

a

bou0

1

inf1110

o

q

q

### testn82.txt

c

l

q

e

o

bou0

q

c

i

1

inf1110

q

q

### testl1.txt

c

c

inf0001

1

30

jeudi 15:00

1

c

inf0002

1

30

jeudi 15:00

1

… Suppresion des données redontantes (Création de 96 cours.) …

c

inf0098

1

30

jeudi 15:00

1

c

inf0099

1

30

jeudi 15:00

1

q

q

### testl2.txt

e

c

001

Yann

c

002

Yann

… Suppresion des données redontantes (Création de 96 dossiers.) …

c

099

Yann

c

100

Yann

q

q

### testl3.txt

e

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

c

Bourdeau

Yann

### testl4.txt

c

c

inf1110

1

30

jeudi 15:00

1

q

e

c

001

Yann

… Suppresion des données redontantes (Création de 28 dossiers.) …

c

030

Yann

a

0010

1

inf1110

o

… Suppresion des données redontantes (Création de 28 inscriptions.) …

a

0300

1

inf1110

o

q

q

### testh1.txt

c

c

inf1110

1

30

jeudi 15:00

1

c

inf1110

1

n

q

q

### testh2.txt

c

c

inf1110

1

30

jeudi 15:00

1

q

e

c

Bourdeau

Yann

a

bou0

1

inf1110

o

a

bou0

1

inf1110

o

q

q

### testh3.txt

c

a

1

inf1110

n

q

q

### testh4.txt

e

c

Bourdeau

Yann

q

c

c

INF1110

1

30

Jeudi 15:00

1

q

e

u

bou0

1

inf1110

q

q

### testh5.txt

e

o

bou0

n

q

q

### testh6.txt

e

c

Bourdeau

Yann

a

bou0

1

inf1110

n

q

q

### testh7.txt

c

i

1

inf1110

n

q

q

### testh8.txt

c

c

INF1110A

1NF1110

INF111A

inf1110

4

1

a

30

jeudi 15:00

4

1

q

q

### testh9.txt

c

c

INF1110

1

30

jeudi 15:00

1

a

4

1

INF1110a

1NF1110

INF111A

INF1110

z

n

q

q