TD-TP n° 10 P_PILE

Thème : TD (2 h ½) débouchant ensuite sur un TP (1 heure ½). Ils portent sur le concept de T.A.D. (type abstrait de données) à concrétiser en cours n° 10. Introduction aux Unbounded.
Concepts : Réalisation d'une **pile** la plus large et la plus "puissante" possible (outil très utile pour certains TD-TP).
Méthode de travail : On examinera plusieurs approches et réalisations de « l'objet » en question (ici une pile). On critiquera, avec l'enseignant, chaque proposition de réalisation pour arriver par améliorations successives à un « objet » ou T.A.D. acceptable. Cet **outil sera utile pour d'autres travaux à venir**.
définition : une **pile** est un objet complexe permettant de sauvegarder des éléments et de les restituer suivant le principe LI-FO c'est-à-dire dernier entré-premier servi (**L**ast **I**n-**F**irst **O**ut), c'est le contraire d'une file (FI-FO) premier entré-premier servi. Il est clair que nous aurons là l'exemple classique d'un type abstrait de données (cf. cours n°10). Nous allons y arriver par étapes successives (on aura soin de ne pas brûler les étapes ¼ heure, environ, à chacune des 5 questions).

PROBLEME : soit à réaliser une pile dédiée au type Character (pour commencer et pour fixer les idées) mais plus loin on aura envie de généraliser à d'autres types de données.

0°) première réalisation de cette pile : (qu'en pensez-vous?)

```
procedure Pile_0 is
type T_Vect is array (Natural range <>) of Character;
Max_Pile : constant := 120;  -- constante universelle
subtype T_Ind is Natural range 0..Max_Pile;
subtype T_Pile is T_Vect (1..Max_Pile);

Ma_Pile  : T_Pile;
Sommet   : T_Ind := 0;
-- deux variables du P.P. mais globales aux deux procédures ci-dessous!!

Carcou, Precedent : Character; -- pour le P.P
    procedure Empiler (C : in Character) is
    begin
        Sommet := Sommet  + 1;
        Ma_Pile (Sommet) := C;
    end Empiler;
    function Depiler return Character is
    begin
        Sommet := Sommet - 1;
        return Ma_Pile (Sommet + 1);
    end Depiler;
begin
    --........
    -- Lire (Carcou);
    Empiler (Carcou);
    --........
    -- Carcou := Ma_Pile(40);
    Precedent := Depiler;
    --.......
end Pile_0;
```

Relevez tout ce qui vous pose problème, ne cherchez pas d'erreurs de syntaxe !

1°) Première amélioration (qu'en pensez-vous?)

On décide (voir critiques précédentes) de créer un paquetage permettant de fabriquer (concevoir et réaliser) un véritable outil accessible après l'avoir évoqué avec **`with`**.

```
package P_Pile_1 is -- spécifications dans un fichier .ads

    procedure Empiler (C : in Character);
    function Depiler return Character;

end P_Pile_1;
```

```
package body P_Pile_1 is -- réalisation dans un fichier .adb

type T_Vect is array (Natural range <>) of Character;
Max_Pile : constant := 120;

subtype T_Ind is Natural range 0..Max_Pile;
subtype T_Pile is T_Vect (1..Max_Pile);

La_Pile : T_Pile;
Sommet  : T_ind; -- mis à 0 dans le bloc initialisation!
-- variables globales aux sous-programmes mais internes au paquetage

    procedure Empiler (C : in Character) is
    begin
        Sommet := Sommet + 1;
        La_Pile(Sommet) := C;
    end Empiler;

    function Depiler return Character is
    begin
        Sommet := Sommet - 1;
        return La_Pile (Sommet + 1);
    end Depiler;

begin -- bloc d'initialisation du paquetage
    Sommet := 0;
end P_Pile_1;
```

Imaginez un petit programme qui utiliserait ce paquetage. Ajoutez les contraintes d'utilisation à votre critique.

```
with ? ;
use ? ;
procedure Ts_Pile is
?
begin
?
end Ts_Pile;
```

Remplacez les ?

2°) deuxième amélioration (qu'en pensez-vous?)

Gros progrès (généricité et plus de fonctionnalités). Est-ce suffisant ?

```
generic
    type T_Elem is private;
    Max_Pile    : in Natural;
package P_Pile_2 is
    procedure Empiler (Elem : in T_Elem);
    function Depiler return T_Elem;
    function Pile_Vide return Boolean;
    function Pile_Pleine return Boolean;
    Exc_Pile_Erreur : exception;
end P_Pile_2;
```

spécifications

```
procedure Test is -- programme d'utilisation
......
      package P_Entier is new P_Pile_2 (Max_Pile =>200, T_Elem => Integer);
      package P_Les_Date is new P_Pile_2 (T_Date,50);
......
begin
……..
end Test;
```

Deux instanciations !

Imaginez des utilisations

```
package body P_Pile_2 is -- réalisation
  type T_Vect is array (Natural range <>) of T_Elem;
  subtype T_Ind is Natural range 0..Max_Pile;
  subtype T_Pile is T_Vect (1..Max_Pile);

  La_Pile : T_Pile;
  Sommet  : T_Ind;

    procedure Empiler (Elem : in T_Elem) is
    begin
        Sommet := Sommet + 1;
        La_Pile (Sommet) := Elem;
        exception
            when others => raise Exc_Pile_Erreur;
    end Empiler;
    function Depiler return T_Elem is
    begin
        Sommet := Sommet - 1;
        return La_Pile (Sommet+ 1);
        exception
            when others => raise Exc_Pile_Erreur;
    end Depiler;
    function Pile_Vide return Boolean is
    begin
        return Sommet = 0;
    end Pile_Vide;
    function Pile_Pleine return Boolean is
    begin
        return Sommet = Max_Pile;
    end Pile_Pleine;
begin
    Sommet := 0;
end P_Pile_2;
```

Critiques toujours !

3°) troisième amélioration (qu'en pensez-vous?)

```
generic
    type T_Elem is private;
    Max_Pile    : in Natural; -- Max statique de toutes les piles
package P_Pile_3 is
    type T_Pile is private; -- permet de créer des piles

    procedure Empiler (Une_Pile : in out T_Pile;
                       Elem : in T_Elem);
    procedure Depiler (Une_Pile : in out T_Pile;
                      Elem : out T_Elem);
    -- attention maintenant Depiler est une procédure !!!!
    -- cherchez pourquoi!

    function Pile_Vide (Une_Pile : in T_Pile) return Boolean;
    function Pile_Pleine (Une_Pile : in T_Pile) return Boolean;
    Exc_Pile_Erreur : exception;
    private
        type T_Vect is array (Natural range <>) of T_Elem;
        subtype T_Ind is Natural range 0..Max_Pile;

        type T_Pile is record -- Sommet et Pile indissociables
           Sommet : T_Ind := 0;
           V      : T_Vect (1..Max_Pile);
        end record;
end P_Pile_3;
```

Le programme d'utilisation s'impose évidemment pour mieux comprendre !

```
package body P_Pile_3 is
    procedure Empiler (Une_Pile : in out T_Pile;
                       Elem : in T_Elem) is
    begin
        Une_Pile.Sommet := Une_Pile.Sommet + 1;
        Une_Pile.V (Une_Pile.Sommet) := Elem;
        exception
            when others => raise Exc_Pile_Erreur;
    end Empiler;

    procedure Depiler (Une_Pile : in out T_Pile;
                      Elem : out T_Elem) is
    begin
        Une_Pile.Sommet := Une_Pile.Sommet - 1;
        Elem := Une_Pile.V (Une_Pile.Sommet + 1);
        exception
            when others => raise Exc_Pile_Erreur;
    end Depiler;

    function Pile_Vide (Une_Pile : in T_Pile) return Boolean is
    begin
        return Une_Pile.Sommet = 0;
    end Pile_Vide;

    function Pile_Pleine(Une_Pile: in T_Pile) return      Boolean is
    begin
        return Une_Pile.Sommet = Max_Pile;
    end Pile_Pleine;
end P_Pile_3;
```

4°) quatrième amélioration (qu'en pensez-vous ? Mieux ! Parfait ?)

```
generic
    type T_Elem is private;
    Max_Pile : in Natural;
package P_Pile_4 is
    subtype T_Max is Natural range 0..Max_Pile;
    type T_Pile (Le_Max : T_Max) is limited private;
    procedure Empiler (Une_Pile : in out T_Pile;
         Elem : in T_Elem);
    procedure Depiler (Une_Pile : in out T_Pile;
                       Elem : out T_Elem);
    function Pile_Vide (Une_Pile : in T_Pile) return Boolean;
    function Pile_Pleine (Une_Pile : in T_Pile) return Boolean;

    Exc_Pile_Pleine, Exc_Pile_Vide,
    Exc_Pile_Erreur : exception;
    private
        type T_Vect is array (T_Max range <>) of T_Elem;
        type T_Pile (Le_Max : T_Max) is record
           Sommet : T_Max := 0; -- en fait dans l'intervalle 0..Le_Max
           V      : T_Vect (1..Le_Max);
        end record;
end P_Pile_4;
```

Encore des critiques ?

```
package body P_Pile_4 is
    procedure Empiler (Une_Pile : in out T_Pile;
                       Elem : in T_Elem) is
    begin
        if Une_Pile.Sommet = Une_Pile.Le_Max
        then raise Exc_Pile_Pleine; end if;
        Une_Pile.Sommet := Une_Pile.Sommet + 1;
        Une_Pile.V (Une_Pile.Sommet) := Elem;
    end Empiler;
    procedure Depiler (Une_Pile : in out T_Pile;
                       Elem : out T_Elem) is
    begin
        if Une_Pile.Sommet = 0
        then raise Exc_Pile_Vide;
        end if;
        Une_Pile.Sommet := Une_Pile.Sommet - 1;
        Elem := Une_Pile.V (Une_Pile.Sommet + 1);
    end Depiler;
    function Pile_Vide (Une_Pile : in T_Pile) return Boolean is
    begin
        return Une_Pile.Sommet = 0;
        exception
            when others => raise Exc_Pile_Erreur;
    end Pile_Vide;
    function Pile_Pleine (Une_Pile : in T_Pile) return Boolean is
    begin
        return Une_Pile.Sommet = Une_Pile.Le_Max;
        exception
            when others => raise Exc_Pile_Erreur;
    end Pile_Pleine;
end P_Pile_4;
```

5°) ultime amélioration (mais c'est à vous de travailler …. Surtout en TP !)

Partez de la dernière solution et ajoutez les deux fonctionnalités `Haut_Pile` et `Hauteur` qui offrent une vue sur le "haut de la pile" (respectivement sur la dernière valeur empilée et sur la hauteur de la pile) mais attention : la pile n'est pas perturbée par ces fonctionnalités. Ajoutez aussi une procédure `Vider_Pile` (qui dit bien ce qu'elle veut dire !).Vous supprimez l'exception superflue. Vous remplacerez les instructions (de levée d'exceptions) `raise` par des `Raise_Exception`.

TP Pile

1. Créez le répertoire tp10. Copiez les deux fichiers p_pile_4.squ (ads et adb).
2. Réalisez la pile définitive. Partez du paquetage P_Pile_4 (Fichiers renommés p_pile_4.ads et p_pile_4.adb). Apportez les améliorations demandées : `Haut_Pile`, `Hauteur`, `Vider_Pile` et les `Raise_Exception`). Compilez ! Mais il faut tester ! (cf. `Ts_Pile`)

Exercice : pour tester le paquetage (programme `Ts_Pile`). Un ruban (cf. algorithmique) mais en fait on parlera ici de fichier, est censé contenir « une ou des » phrase(s) « palindrome ».

Exemples (de phrases palindromes) :

- Tu l'as trop écrasé, César, ce port salut.
- Ésope reste ici et se repose.
- Noël à Léon.
- La malade pédala mal.
- Élu par cette crapule.

En connaissez-vous d'autres ?

Seuls les caractères lettres seront pris en compte ! (on négligera les caractères parasites tels que les points, les virgules, les espaces etc. Les accents seront supprimés ! On ne distinguera pas les majuscules des minuscules. Il y a des filtres à mettre en place ! On pourrait créer beaucoup de petits fichiers les uns n'étant pas des phrases palindromes, d'autres étant au moins quelques uns des cinq textes ci dessus. Pour gagner du temps on aura **un seul fichier** d'identité système : "`pile.in`" (**à fabriquer**) dont **chaque ligne** sera (ou ne sera pas) une phrase palindrome.

Algorithme de `ts_pile` (**pour chaque ligne**) :
1) On lit la phrase (comme une ligne d'un fichier) dans un `Unbounded_String`.
2) Pour chaque caractère de la ligne lue, s'il est alphabétique et après réduction en minuscule non accentuée (filtrage) on le stocke (deux fois) :

- D'une part : dans un **autre** `Unbounded_String` (paquetage `Ada.Strings.Unbounded`)
- D'autre part : dans une PILE de caractères (issue d'une instanciation de P_PILE_4)

3) A la fin on compare le `Unbounded_String` et la PILE (**élément par élément**) puis on édite un texte indiquant si c'est (ou non) une phrase palindrome.
Voir et copier le fichier `ts_pile.adb.squ`, **renommez** en `ts_pile.adb` et complétez.
Rendez (avec une entête !) les listings des 2 fichiers `.adb` et le fichier `pile.in`.

Conservez, bien au chaud, le paquetage `p_pile_4` ; on s'en servira bientôt !

```
-- fichier p_pile_4.ads.squ à renommer p_pile_4.ads
-- insérer votre entête
-- ajoutez les spécifications des 3 fonctionnalités demandées
-- réalisez le body et testez
-- T. Avignon D. Feneuille Oct. 2000
-- pour TD - TP 10
--

generic
   type T_Elem is private;
   Max_Pile : in Natural;
   -- le max de toutes les piles pour un T_ELEM

package P_Pile_4 is

   subtype T_Max is Natural range 0..Max_Pile;

   type T_Pile
         (Le_Max : T_Max) is limited private;
   -- limited interdit l'affectation

   procedure Empiler (
         Une_Pile : in out T_Pile;
         Elem     : in     T_Elem  );

   procedure Depiler (
         Une_Pile : in out T_Pile;
         Elem     :    out T_Elem  );

   function Pile_Vide (
         Une_Pile : in     T_Pile )
     return Boolean;

   function Pile_Pleine (
         Une_Pile : in     T_Pile )
     return Boolean;

   Exc_Pile_Pleine,
      Exc_Pile_Vide,
      Exc_Pile_Erreur  : exception;

private

   type T_Vect is array (T_Max range <>) of T_Elem;

   type T_Pile
         (Le_Max : T_Max) is
      record
         Sommet : T_Max := 0;
       -- en fait entre 0..LE_MAX
         V      : T_Vect (1 .. Le_Max);
      end record;

end P_Pile_4;
```

```
-- fichier p_pile_4.adb.squ à renommer p_pile_4.adb
-- mettez votre entête
-- réalisez les 3 fonctionnalités manquantes
-- supprimez l'exception superflue
-- comprenez bien les écritures des Raise_Exception
-- D.Feneuille & T.Avignon   Octobre 2000
-- pour TD-TP 10

with Ada.Exceptions;
use  Ada.Exceptions;
package body P_Pile_4 is
   procedure Empiler (
         Une_Pile : in out T_Pile;
         Elem     : in     T_Elem  ) is
   begin
      if Une_Pile.Sommet = Une_Pile.Le_Max
            then
         Raise_Exception ( Exc_Pile_Pleine'Identity, "Empiler ");
      end if;
      Une_Pile.Sommet := Une_Pile.Sommet + 1;
      Une_Pile.V (Une_Pile.Sommet) := Elem;
   end Empiler;

   procedure Depiler (
         Une_Pile : in out T_Pile;
         Elem     :    out T_Elem  ) is
   begin
      if Une_Pile.Sommet = 0
            then
         Raise_Exception ( Exc_Pile_Vide'Identity, "Dépiler ");
      end if;
      Une_Pile.Sommet := Une_Pile.Sommet - 1;
      Elem := Une_Pile.V (Une_Pile.Sommet + 1);
   end Depiler;

   function Pile_Vide (
         Une_Pile : in     T_Pile )
     return Boolean is
   begin
      return Une_Pile.Sommet = 0;
   exception
      when others =>
         Raise_Exception( Exc_Pile_Erreur'Identity, "Pile Vide ");
   end Pile_Vide;

   function Pile_Pleine (
         Une_Pile : in     T_Pile )
     return Boolean is
   begin
      return Une_Pile.Sommet = Une_Pile.Le_Max;
   exception
      when others =>
         Raise_Exception( Exc_Pile_Erreur'Identity, "Pile Pleine ");
   end Pile_Pleine;
end P_Pile_4;
```

```
-- fichier ts_pile.adb.squ à renommer ts_pile.adb
-- insérez votre entête, finissez et testez !!
-- un fichier texte (voir donc TEXT_IO) contient (ou non) des
-- phrases palindrome. Une ligne = une phrase.
-- algorithme : ouvrir le fichier (voir cours fichier textes)
-- "dérouler le fichier jusqu'à fin de fichier" (cours fichier textes)
-- on lit une ligne dans un Unbounded_String que l'on va exploiter.
-- pour chaque caractère lu, s'il est lettre le rendre minuscule et non
-- accentué voir le paquetage Ada.Characters.Handling
-- le "ranger" dans la PILE d'une part (empiler) et dans un Unbounded
-- comparer la pile et le Unbounded_String
-- (depiler et element). Aller seulement à moitié !
-- Attention une phrase vide (après filtrage) n'est pas un palindrome.
--
with Ada.Text_Io, P_Pile_4, Ada.Characters.Handling,
   Ada.Strings.Unbounded.Text_Io;
with Ada.Exceptions;

procedure Ts_Pile is

   -- instanciation d'un vrai paquetage P_Pile :
   package P_Pile is new P_Pile_4 (Character,500);
   use Ada.Text_Io, P_Pile,
   Ada.Characters.Handling,
   Ada.Strings.Unbounded,
   Ada.Strings.Unbounded.Text_Io;
   use Ada.Exceptions;

   -- déclarations des objets utiles
   Fich_In : File_Type;
   Ligne   : Unbounded_String; -- pour lire la phrase
   Chaine  : Unbounded_String; -- pour stocker la phrase filtrée
   La_Pile : T_Pile (100);  -- pour stocker aussi la phrase filtrée
   -- ajoutez d'autres initialisations !!

begin
   Open (Fich_In, In_File, "pile.in"); -- ouvrir le fichier
   loop
      exit when End_Of_File (Fich_In); -- boucle jusqu'à la fin
      Ligne := Get_Line(Fich_In); -- lire une ligne
      -- préparez ici les initialisations (important!!)
      --
      for Ind in 1..Length(Ligne) loop
      -- faire un filtre et stocker dans La_Pile et dans Chaine
      end loop;
      -- comparez La_Pile et Chaine autre boucle loop
      -- et établissez un bilan pour la ligne lue
      --
   end loop;
   Close (Fich_In);
exception
   when Marqueur: others =>
      Put_Line(Exception_Information(Marqueur));
end Ts_Pile;
```

Corrigés du TD mais non exhaustifs !

Les critiques les plus sévères (question par question) :

0°) a) aucune spécification
b) tout est dans tout (S/P et P.P.). On a une vue sur tout. On peut accéder au cœur de la pile! (telle Carcou := Ma_Pile (40); ⇒ incohérent)
c) variables globales = danger (rappel !)
d) pas assez de fonctionnalités !
e) c'est un peu de maquettage (sans plus) ⇒ klennex (à jeter) Un paquetage s'impose !

1°) a) avec un paquetage on a une meilleure spécification (vue uniquement sur Empiler et Depiler)
b) la taille maximum est statique, et bien sûr on est restreint au type Character.
c) manque toujours de fonctionnalités
d) le paquetage (qui n'est plus le P.P.) exporte un unique objet pile (impossible d'avoir deux piles !)

2°) a) il y a un peu plus de fonctionnalités (état de la pile "pleine et vide")
b) la généricité augmente le dynamisme de l'outil
c) la structure étant toujours encapsulée dans le body on n'a hélas qu'une pile par paquetage instancié.
d) à chaque instanciation il y a création d'une exception (le nom est identique!) donc un **use** ne servira à rien (il faudra utiliser la notation pointée!) ainsi que pour Pile_Vide et Pile_Pleine.

3°) a) on peut créer plusieurs objets pile grâce au type T_Pile (mais même taille max et même élément)
b) une petite pile devra avoir une taille maxi égale à la plus grande (dommage !). Prendre un exemple.
c) la structure (ici un tableau) si elle est "visible" par l'utilisateur est bien inaccessible (privé !)
d) la fonction est devenue procédure car il faut préciser la pile concernée (**in out** obligatoire)
e) exception trop vague (non personnalisée)
f) le **private** permet l'affectation (très incohérent pour une structure de pile)

4°) a) les spécifications pourraient être plus documentées (exception notamment !)
b) une des exceptions d'ailleurs a-t-elle encore un sens ?
c) chaque pile (à sa déclaration) a bien sa taille maxi propre mais pourrait-on s'en affranchir ?
d) d'autres fonctionnalités (mais pas trop !) Haut_Pile, Hauteur, Vider_Pile à voir !.

5°) C'est le TP à rendre ! Sera mis en corrigé plus tard !