

```

with Ada.Text_IO, Files;
procedure Test_Files is

    type Type_Appareil is ( __Planeur,
        _____Cargo,
        _____Commercial,
        _____Chasseur,
        _____Reconnaissance);
    package Appareil_Io is new Ada.Text_IO.Enumeration_IO ( Type_Appareil );

    package Naturel_Io is new Ada.Text_IO.Integer_IO ( Natural );

    package File_Appareil is new Files ( Élément => Type_Appareil);

File_Pour_La_Piste : File_Appareil.File;
File_Initiale : File_Appareil.File;

    procedure Imprimer ( La_File : in File_Appareil.File) is separate;
    function "=" (Gauche _ : in File_Appareil.File;
        _____Droite__ : in File_Appareil.File) return Boolean
        _____renames File_Appareil.Est_Égal;
begin
File_Appareil.Purger ( File_Pour_La_Piste);
for Index in Type_Appareil loop
    File_Appareil.Ajouter ( Index, À_La_File => File_Pour_La_Piste);
end loop;
File_Appareil.Copier ( File_Pour_La_Piste, File_Initiale);

```

```
Imprimer (File_Pour_La_Piste);
File_Appareil.Enlever ( File_Pour_La_Piste);
File_Appareil.Enlever ( File_Pour_La_Piste);
File_Appareil.Enlever ( File_Pour_La_Piste);
Ada.Text_IO.Put ( " La tête de la file est :");
Appareil_IO.Put (File_Appareil.Tête_De (File_Pour_La_Piste));
Ada.Text_IO.New_Line;
Ada.Text_IO.Put ( " La longueur de la file est : ");
Naturel_IO.Put ( File_Appareil.Longueur_De (File_Pour_La_Piste), width => 2);
File_Appareil.Purger ( File_Pour_La_Piste);
Ada.Text_IO.New_Line;
Ada.Text_IO.Put ( " La longueur de la file après la purge est : ");
Naturel_IO.Put ( File_Appareil.Longueur_De (File_Pour_La_Piste), width => 2);
Ada.Text_IO.New_Line;
if File_Pour_La_Piste /= File_Initiale then
    Ada.Text_IO.Put_Line (" La File_Pour_La_Piste est différente de la
File_Initiale");
end if;
end Test_Files;
```

```
separate ( Test_Files)
procedure Imprimer ( La_File : in File_Appareil.File) is
  L_Appareil : File_Appareil.Pour_Itérer.Itérateur;
begin
  File_Appareil.Pour_Itérer.Initialiser ( L_Appareil, File_Pour_La_Piste);
  Ada.Text_Io.Put_Line ( " Les appareils en file pour la piste sont : ");
  while not File_Appareil.Pour_Itérer.Est_Fini ( L_Appareil ) loop
    Appareil_IO.Put (File_Appareil.Pour_Itérer.Valeur_De ( L_Appareil ));
    Ada.Text_Io.New_Line;
    File_Appareil.Pour_Itérer.Aller_Suivant ( L_Appareil );
  end loop;
end Imprimer;
```

```

generic
type Élément is private;
package Files is
  type Noeud is private;
  type Structure is access Noeud;

  type File _ is limited private;
  procedure Purger __ ( La_File__ : in out File);
  procedure Ajouter __ ( La_Valeur_ : in Élément;
                        _____À_La_File_ : in out File);

  procedure Enlever __ ( La_File__ : in out File);
  procedure Copier_   __ ( De_La_File _ : in File;
                        _____À_La_File_ : in out File);

  function Longueur_De _ ( La_File : in File) return Natural;
  function Est_Vide __ ( La_File : in File) return Boolean;
  function Est_Égal ( Gauche _ : in File;
                    _____ Droite __ : in File) return Boolean;
  function Tête_De __ ( La_File : in File) return Élément;
Débordement, Épuisement_ : exception;
package Pour_Itérer is
  subtype Itérateur is Structure;
  procedure Initialiser _ ( L_Itérateur __ : in out Itérateur;
                          _____ Avec_La_File _ : in File );
  procedure Aller_Suivant ( L_Itérateur_ : in out Itérateur);
  function Valeur_De__( L_Itérateur_ : in Itérateur) return Élément;
  function Est_Fini__( L_Itérateur_ : in Itérateur) return Boolean;
  Erreur_Itérateur : exception;

```

```
    end Pour_Itérer;  
private  
  type File is  
    record  
      La_Tête _: Structure;  
      La_Queue_: Structure;  
    end record;  
  type Noeud is  
    record  
      L_Élément _: Éléments;  
      Suivant _: Structure;  
    end record;  
end Files;
```

```

with Ada.Unchecked_Deallocation;
package body Files is

    package body Pour_Itérer is

        procedure Initialiser _( L_Itérateur __: in out Itérateur;
                                Avec_La_File _: in_File) is

begin
    L_Itérateur := Itérateur (Avec_La_File.La_Tête);
end Initialiser;

        procedure Aller_Suivant ( L_Itérateur _: in out Itérateur) is
begin
    L_Itérateur := Itérateur (L_Itérateur.Suivant);
    exception

        when Constraint_Error =>
            raise Erreur_Itérateur;
end Aller_Suivant;

        function Valeur_De ( L_Itérateur : in Itérateur ) return Élément is
begin
    return L_Itérateur.L_Élément;
    exception
        when Constraint_Error =>
            raise Erreur_Itérateur;
end Valeur_De;

```

```

    function Est_Fini ( L_Itérateur : in Itérateur ) return Boolean is
begin
    return (L_Itérateur = null);
end Est_Fini;

end Pour_Itérer;

procedure Purger _(La_File__ : in out File) is
    Courant, Pointeur_Courant : Pour_Itérer.Itérateur;
    procedure Delete is new Ada.Unchecked_Deallocation ( _Object __=> Noeud ,
        _____Name __=>
Pour_Itérer.Itérateur );
begin
    if La_File.La_Tête /= La_File.La_Queue then
        Pour_Itérer.Initialiser ( Courant, Avec_La_File => La_File);
        while not Pour_Itérer.Est_Fini ( Courant ) loop
            Pointeur_Courant := Courant;--
            Pour_Itérer.Aller_Suivant ( Courant );
            Delete (Pointeur_Courant);
        end loop;
    end if;
    La_File := File'( La_Tête __=> null,
        La_Queue __=> null);
end Purger;

procedure Ajouter (_La_Valeur_ : in Élément;

```



```

À_La_File.La_Queue  _:= À_La_File.La_Tête;
À_L_Index          _____:= À_La_File.La_Tête;
De_L_Index_____ := De_L_Index.Suivant;
while De_L_Index /= null loop
    À_L_Index.Suivant := new Noeud'( L_Élément => De_L_Index.L_Élément,
    _____Suivant _____=> null);
    À_L_Index _____:= À_L_Index.Suivant;
    De_L_Index _____:= De_L_Index.Suivant;
    À_La_File.La_Queue _:= À_L_Index;
end loop;
end if;
exception
    when Storage_Error =>
        raise Débordement;
end Copier;

procedure Enlever ( La_File_: in out File) is
    procedure Delete is new Ada.Unchecked_Deallocation ( _Object __=> Noeud ,
    _____Name __=> Structure );
    À_Enlever : Structure := La_File.La_Tête;
begin
    La_File.La_Tête := La_File.La_Tête.Suivant;
    Delete ( À_Enlever);
    if La_File.La_Tête = null then
        La_File.La_Queue := null;
    end if;
    exception

```

```

        when Constraint_Error _=>
            raise Épuisement;
    end Enlever;

    function Longueur_De (_La_File_: in File) return Natural is
        Compte _: Natural := 0;
        Index _: Structure _ := La_File.La_Tête;
    begin
        while Index /= null loop
            Compte := Compte + 1;
            Index := Index.Suivant;
        end loop;
        return Compte;
    end Longueur_De;

    function Est_Égal ( Gauche _: in File;
                       Droite __: in File) return Boolean is
        Index_Gauche _: Structure := Gauche.La_Tête;
        Index_Droit  _: Structure := Droite.La_Tête;
    begin
        while Index_Gauche /= null loop
            if Index_Gauche.L_Élément /= Index_Droit.L_Élément then
                return False;
            else
                Index_Gauche := Index_Gauche.Suivant;
                Index_Droit := Index_Droit.Suivant;
            end if;
        end loop;
    end Est_Égal;

```

```
    end loop;
    return (Index_Droit = null);
    exception
        when Constraint_Error =>
            return False;
end Est_Égal;

function Est_Vide ( La_File : in File) return Boolean is
begin
    return (La_File.La_Tête = null);
end Est_Vide;

function Tête_De ( La_File : in File) return Élément is
begin
    return La_File.La_Tête.L_Élément;
    exception
        when Constraint_Error =>
            raise Épuisement;
end Tête_De;

end Files;
```