Algorithmique & programmation

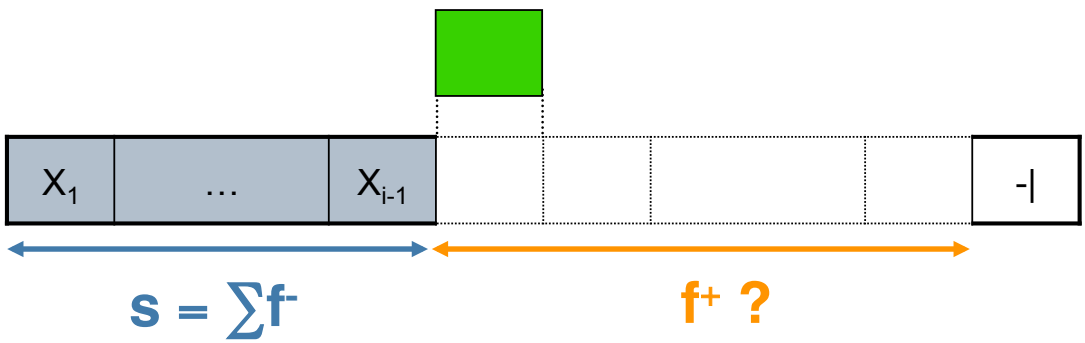
Chapitre 3 : Fichiers séquentiels

Algorithme traitant un seul fichier

Premiers algorithmes

Somme des éléments d'un fichier

- On suppose que f est un fichier d'entiers non vide
- Vers l'hypothèse
  - À un instant donné, on a fait la somme des éléments d'un sous-fichier déjà examiné
  - En schématisant



Somme des éléments d'un fichier

- Raisonnement par récurrence

Hypothèse $s = \Sigma f^-$

➤ $f^+ = <>$ ⇨ $s = \Sigma f$ ⇨ résultat = s ✳

➤ $f^+ \neq <>$ ⇨ lire (f , c) ; s := s + c ; ➡ ***H***

Itération tantque non fdf (f) faire...

Initialisation relire (f) ; lire (f , s) ; ➡ ***H***

fonction **somme** d'un fichier d'entiers

fonction somme (d f : fichier deentier) : entier;
spécification $\{f \neq <>\}$ ➔ $\{résultat = \Sigma f\}$
    s : entier ; c : entier ;
debproc
    $\{f \neq <>\}$
    relire (f) ;
    $\{f = <>, f^+ = f\ ,\ \neg\ fdf\ (f)\}$
    lire (f , s) ;
    $\{s = x_1\ , f = x_1, s = \Sigma f\}$
    tantque non fdf (f) **faire**
        lire (f , c) ; $\{s = \Sigma f^-\}$
        s := s + c ; $\{s = \Sigma f^- + c\}$
        $\{s = \Sigma f\}$
    finfaire ;
    $\{s = \Sigma f\ , f = f)$ ➔ $s = \Sigma f\}$
    retour s ;
finproc ;

fonction **Somme** (fichier d'entiers)

```
-- Importer le module générique Sequential_io, use inutile
with Sequential_Io;

package P_Fentier is
   -- Instancier le paquetage Sequential_Io
   -- Pour manipuler des fichiers de Integer
   package P_Entier_Io is new Sequential_Io(Integer);
   use P_Entier_Io;

   -- Fonctions manipulant un fichier d'Entiers non vide
   function SommeNonVide(F : in P_Entier_Io.File_Type )
                                                  return Integer;
   function MaxNonVide(F : in P_Entier_Io.File_Type )
                                                  return Integer;
   -- Fonction manipulant un fichier d'Entiers vide
   function SommeVide(F : in P_Entier_Io.File_Type )
                                                  return Integer;
end P_Personne;
```


fonction **SommeNonVide** (fichier d'entiers)

```
package body P_Entier is

   function SommeNonVide (F : in P_Entier_Io.File_Type)
                                                  return integer is
      -- variables pour lire un élément & faire la somme
      val : integer ; s : integer ;
   begin
      reset(F);  -- on se positionne en début de fichier
      read(F, s); -- initialisation de la somme
      while not End_Of_File(F) loop
         -- tant que la fin de fichier n'est pas atteinte…
         read(F, val); --lecture d'un entier dans le fichier
         s := s + val; --ajout de la valeur lue à la somme
      end loop;
      return s ; --retourner la somme
   end SommeNonVide;

   …
end P_Entier;
```

fonction **somme**

- Cas du fichier vide
  - Il faut modifier l'algorithme, **on n'a pas le droit de faire lire (f, s)**
  - On convient que la somme est égale à 0

fonction somme (d f : fichier deentier) : entier;
spécification *{Ø}* ➔ *{résultat = Σf}*
    s : entier ; c : entier ;
debproc
    relire (f) ;
    s := 0 ;
    tantque non fdf (f) **faire**
        lire (f , c) ;
        s := s + c ;
    finfaire ;
    retour s ;
finproc ;

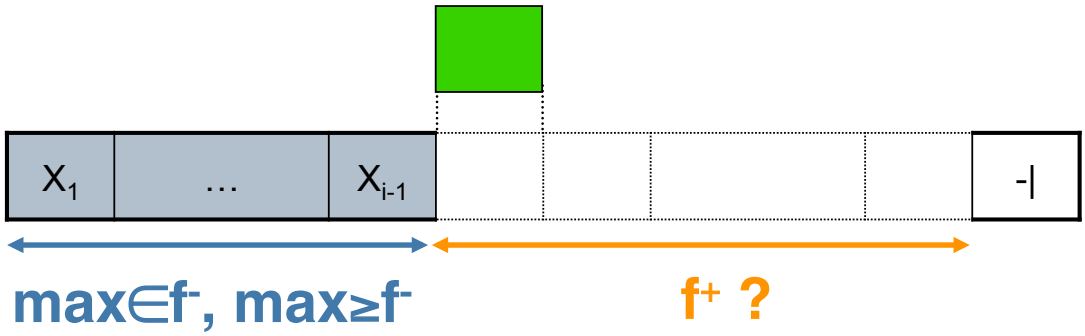
fonction **SommeVide** (fichier d'entiers)

```
package body P_Entier is
  …
  function SommeVide (F : in P_Entier_Io.File_Type)
                                                  return integer is
    -- variables pour lire un élément & faire la somme
    val : integer ; s : integer ;
  begin
    reset(F);  -- on se positionne en début de fichier
    s := 0;    -- initialisation de la somme
    while not End_Of_File(F) loop
      -- tant que la fin de fichier n'est pas atteinte…
      read(F, val); --lecture d'un entier dans le fichier
      s := s + val; --ajout de la valeur lue à la somme
    end loop;
    return s ;      --retourner la somme
  end Affiche;

end P_Entier;
```

Recherche de la valeur maximale

- On cherche une valeur max ∈ f (f non vide), telle que max ≥ f
- Vers l'hypothèse
  - À un instant donné, on a trouvé le max d'un sous-fichier déjà examiné
  - En schématisant



Recherche de la valeur maximale

- Raisonnement par récurrence

Hypothèse max ∈ f^-, max ≥ f^-

➤ $f^+ = <>$ ⇨ *max ∈ f , max ≥ f*
⇨ résultat = max ✳

➤ $f^+ \neq <>$ lire (f , c) ;
⇨ *max ∈ f^- , max ≥ f^- , f = f^- || <c>*
➤➤ max ≥ c ⇨ *rien à faire* ➡ ***H***
➤➤ max < c ⇨ max := c ; ➡ ***H***

Itération tantque non fdf (f) faire...

Initialisation relire (f) ; lire (f , max) ; ➡ ***H***

fonction **maxi**

fonction maxi (d f : fichier de t) : t ;
spécification $\{f \neq <>\} \rightarrow \{résultat = max\ ,\ max \in f\ ,\ max \geq f\}$
    max , c : t ;
debfonc
    relire (f) ;
    lire (f , max) ;
    $\{max \in f\ ,\ max \geq f\}$
    tantque non fdf (f) **faire** $\{max \in f\ ,\ max \geq f\}$
        lire (f , c) ;
        $\{max \in f^-, max \geq f^-, f = f^- // <c>\}$
        si max < c **alors**
            max := c ;
        finsi ; $\{max \in f\ ,\ max \geq f\}$
    finfaire ;
    $\{(fdf(f),\ max \in f\ ,\ max \geq f) \rightarrow (max \in f,\ max \geq f)\}$
    retour max ;
finfonc ;

fonction **MaxNonVide** (fichier d'entiers)

```
package body P_Entier is

  function MaxNonVide (F : in P_Entier_Io.File_Type)
                                                  return integer is
    -- variables pour lire un élément & mémoriser max
    val : integer ; max : integer ;
  begin
    reset(F);     -- on se positionne en début de fichier
    read(F, max);  -- initialisation de max
    while not End_Of_File(F) loop
      read(F, val); --lecture d'un entier dans le fichier
      if max < val then -mise à jour éventuelle de max
        max := val;
      end if;
    end loop;
    return s ; --retourner la somme
  end Affiche;
end P_Entier;
```