Algorithmique & programmation

Chapitre 3 : Fichiers séquentiels

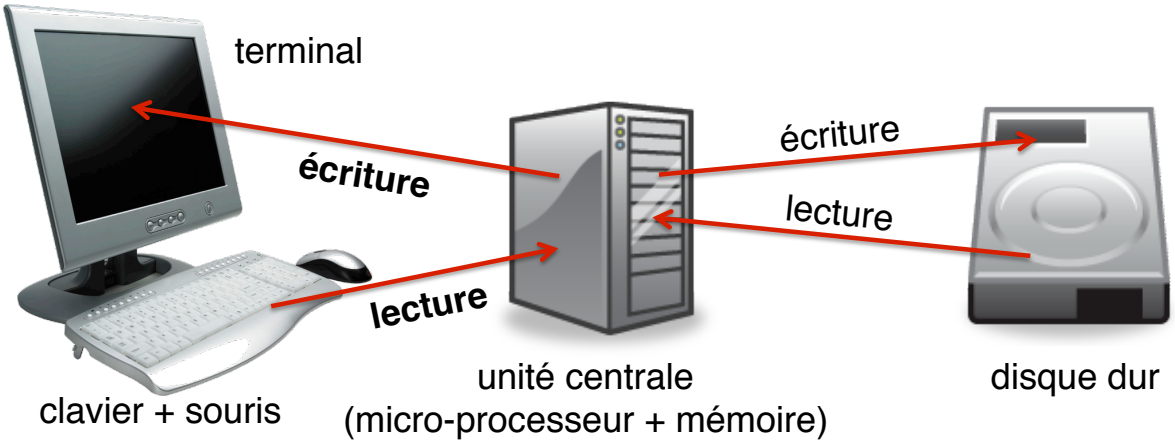
Notion

Définitions

Primitives

Entrées/Sorties

- Les entrées-sorties permettent à l'unité centrale de communiquer avec les périphériques :
  - disques durs,
  - terminaux,
  - ...



Notion de fichier

- Analogie avec
  - suite d'enregistrements sur une bande magnétique
- On parle de support **persistant**...
- Par opposition à la mémoire centrale qui ne peut garder les données que pendant une durée limitée : le temps d'exécution du programme
- Le contenu d'un vecteur, stocké en mémoire centrale, est ainsi perdue lorsque le programme qui l'a créé s'achève

Entrées/Sorties et fichiers

- On distingue les fichiers selon les deux propriétés suivantes :
  - leur **mode de représentation**, c'est-à-dire la nature du codage utilisé pour représenter les informations stockées dans le fichier
  - leur **mode d'accès**, qui décrit la façon selon laquelle on accède à ces informations

Modes de Représentation

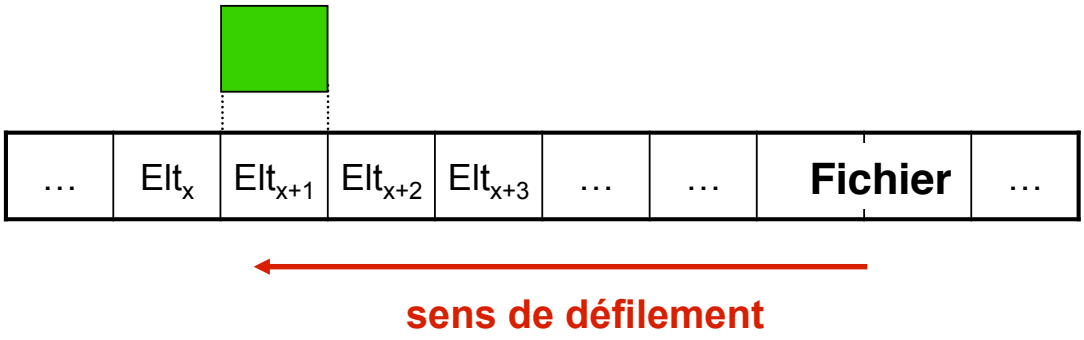
- 2 modes de représentation :
  - les fichiers « **texte** » :
    - Contiennent de l'information codée en ASCII
    - Peuvent être manipulés par un éditeur de texte
    - Facilement lisibles par un opérateur humain
  - les fichiers « **binaires** » (non textuels)
    - l'information n'est pas codée en ASCII
    - Nécessitent un décodage pour pouvoir être "lus" par un opérateur humain
    - Exemple : fichiers exécutables produits par un compilateur

Modes d'accès

- Deux modes d'accès aux fichiers :
  - **accès séquentiel**
    - Les éléments sont accessibles les uns à la suite des autres à partir du premier *(comme en algo)*
  - **accès direct**
    - Les éléments sont immédiatement accessibles par leur position dans le fichier ou **rang**
    - En Ada, le premier élément a le rang 1
    - Cette organisation est analogue à la notion de vecteur si ce n'est que le support de stockage est le disque dur au lieu de la mémoire centrale.

Notion de fichier séquentiel

- Un fichier séquentiel peut être vu comme
  - une bande mobile (un ruban)
    - qui se déplace toujours dans le même sens (**sens de défilement**)
    - devant une **tête de lecture/écriture**



Définition formelle

- Soient
  - V : un ensemble homogène (même type) de valeurs du fichier séquentiels
  - P : un ensemble fini, totalement ordonné, des places du fichier séquentiel
- Un fichier séquentiel à valeurs dans V est une application f de P dans V

- Un fichier peu être vide
- Une relation d'ordre peut être définie sur V

Notation

- Un fichier est noté fonctionnellement
  - (P, f, V)
- ou, plus simplement
  - f
- Un fichier est noté ex extension
  - $<x_1, x_2, x_3, \ldots\ldots\ldots\ldots, x_n>$ ($x_i \in V$)
- Exemples
  - f1 = <'A', 'B', 'D', 'D'>
  - f2 = <1, 23, 43, 17>

Sous-fichier : définition

- C'est une restriction de f à un intervalle de P (une suite d'éléments consécutifs)
- Exemple :
  - Si f = <5, 7, 59, 67, -1, 0, 18>
    - <5, 7, 59> , <67, -1, 0, 18>
      - sont des sous-fichiers de f
    - <5, 59, 18>
      - n'est pas un sous-fichier de f
- Tout fichier peut être considéré comme un sous-fichier de lui-même.

Sous-fichier : notation

- f_i^j = restriction de f à l'intervalle [i..j].
  - <xi, ……, xj>
- f_{i+1}^i est un fichier vide
  - de même, f_i^j est vide si i > j
- Le fichier f considéré comme un sous-fichier est noté f_1^n
- Tout fichier vide sera noté < >

Concaténation de deux fichiers

- Loi de composition interne
  - f1 = <x_1, …… , x_n> ,
  - f2 = <y_1, …… , y_p>
  - x_i, $y_j \in V$ alors
  - f = f1 || f2

  = <x_1, ... , x_n, y_1, ... , y_p>
- On peut décomposer tout fichier f en k sous-fichiers tels que f = f1|| f2 || …… || fk
- Si f comporte n éléments, les fi sont vides pour i > n
- La concaténation est associative mais non commutative

Somme

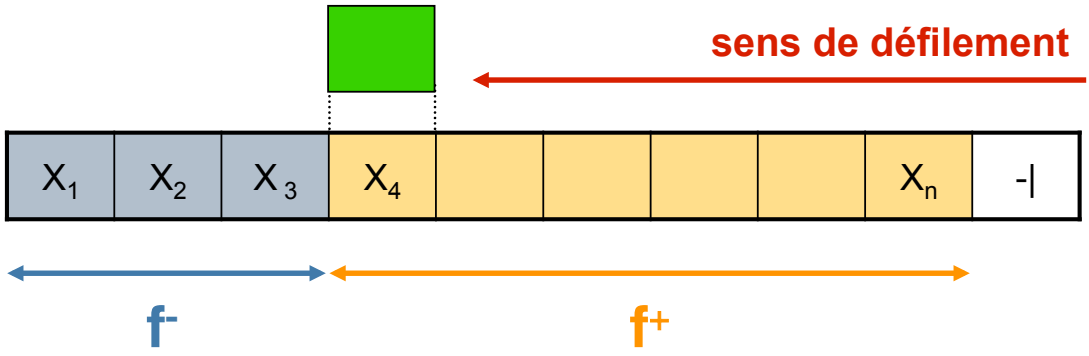
- … d'un fichier et d'une valeur
  - g = f ⊕ val = f ' || <val> || f '', f = f '|| f ''
    - Exemple : f = <5 , 10 , 3 , 9> et val = 12
    - g = f ⊕ val = <5 , 10 , 3 , 12 , 9> ou <5 , 12 , 10 , 3 , 9> ou <12 , 5 , 10 , 3 , 9>…
- … de deux fichiers
  - Soit f, et g = <y_1, y_2,. . . . . y_n>
  - La somme de f et g est définie comme suit :
  - h = f ⊕ y_1 ⊕ y_2 ⊕. . . . . ⊕ y_n

Autres notations

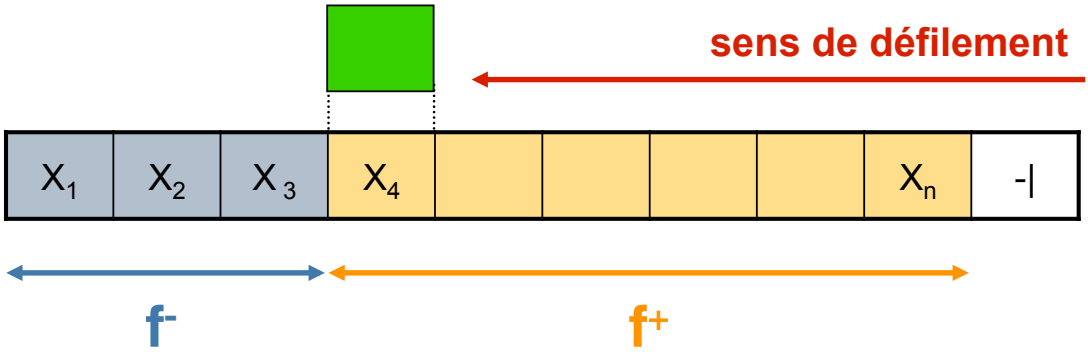
- Soient
  – **f** = <x_1, x_2, …… , x_n>) à valeurs dans un ensemble ordonné V ($x_i \in V$)
  – et **a** une valeur appartenant à V
  - a < f => $\forall i \in [1..n]$, a < x_i
  - $a \in f$ => $\exists i \in [1..n]$, a = x_i
  - $a \notin f$ => $\forall i \in [1..n]$, $a \neq x_i$
  - a = f => $\forall i \in [1..n]$, a = x_i
  - $a \neq f$ => $\forall i \in [1..n]$, $a \neq x_i$
- On peut aussi définir :
  - $a \leq f$, a > f , $a \geq f$.

Actions primitives d'accès

- L'accès à une valeur n'est possible que si l'élément correspondant (élément courant) se trouve en face de la tête de **lecture/écriture**
- Une marque de fin de fichier (notée **-|**) se trouve systématiquement à la fin du fichier et permet d'en détecter la fin



Actions primitives d'accès



- Définitions
  - f = f^- || f^+
  - f^- : sous-fichier déjà « lu »
  - f^+ : sous-fichier qui reste à lire
  - Élément courant = premier élément de f^+ (x_4)

Prédicat fin de fichier

en ada	dans un algorithme
`end_of_file(nom_fichier)`	fdf(nom de fichier)

- fonction booléenne (prédicat) : vrai si et seulement si on a atteint la marque de fin de fichier

- C'est l'exécution des primitives d'accès au fichier en écriture et en lecture qui permet de donner une valeur au prédicat **fin de fichier**

- En-tête

  fonction fdf(d f : fichier de t) : booléen ;

  spécification $\{\ \} \rightarrow \{résultat = (f^+ = <>)\}$

Accès au premier élément en lecture

- Pour avoir accès au premier élément d'un fichier, il faut positionner la tête de **lecture/ écriture** en face de ce premier élément

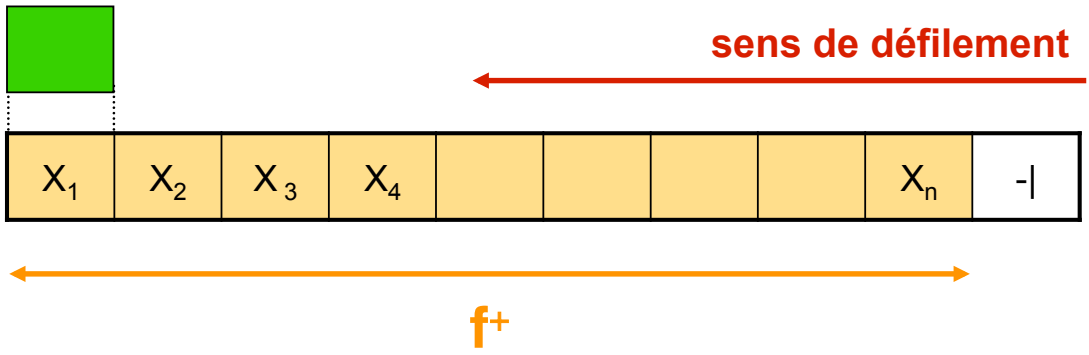
en ada	dans un algorithme
`reset(nom_fichier)`	relire(nom de fichier)

- En-tête :

  procédure relire (d f : fichier de t) ;

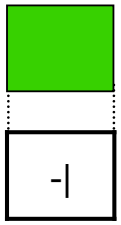
  spécification $\{\ \} \rightarrow \{(\neg fdf(f), f = <>, f^+ = f, f \neq <>) \vee (fdf(f), f = <>, f^+ = <>)\}$

procédure **relire-reset** (fichier non vide)



- f^- est vide
- $f^+ = f$
- l'élément courant = x_1
- **fdf(f)** est faux (on écrit ¬ **fdf(f)**)

procédure **relire-reset** (fichier vide)



- f^- est vide
- f^+ est vide
- **il n'y a pas d'élément courant**
- **fdf(f)** est vrai (on écrit **fdf(f)**)

Détection de la fin du fichier

en ada	dans un algorithme
`end_of_file(nom_fichier)`	fdf(nom de fichier)

- Permet d'accéder à la valeur de l'élément courant et de positionner la tête de lecture sur l'élément suivant :

- Deux étapes
  1. copie de l'élément courant (lecture) pour la mettre dans **val**
  2. avancement du ruban d'une position
     - Si la marque de fin de fichier est sélectionnée alors **fdf (nom de fichier)** prend la valeur vrai
     - Sinon **fdf (nom de fichier)** conserve la valeur faux.

Accès à l'élément courant

en ada	dans un algorithme
`read(nom_fichier,valeur)`	lire(nom de fichier, valeur)

- **lire (`read`) ne peut pas être utilisée si l'on a atteint la fin du fichier**
  - *lorsque fdf est vrai avant l'appel de la procédure*

- En-tête :

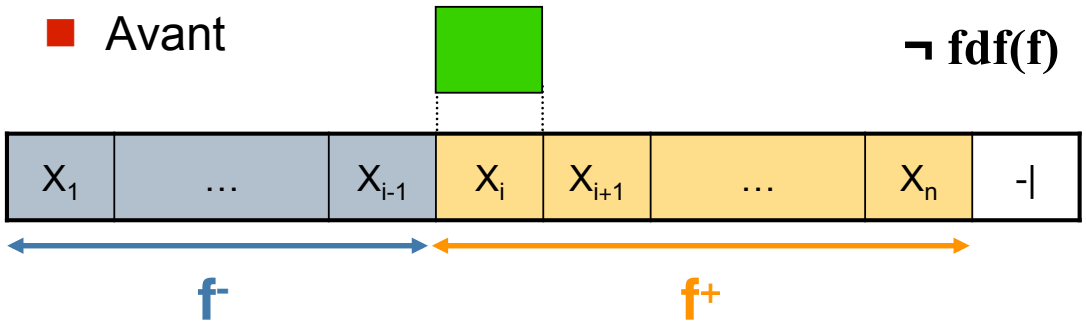
  procédure lire(d f : fichier de t; r val: t);

  spécification $\{f = f_1^{i-1}, f^+ = f_i^n, \neg fdf(f)\} \rightarrow \{val = x_i, f^- = f_1^i, (f^+ = f_{i+1}^n, \neg fdf(f), f^+ \neq <>) \vee (fdf(f), f^+ = <>)\}$

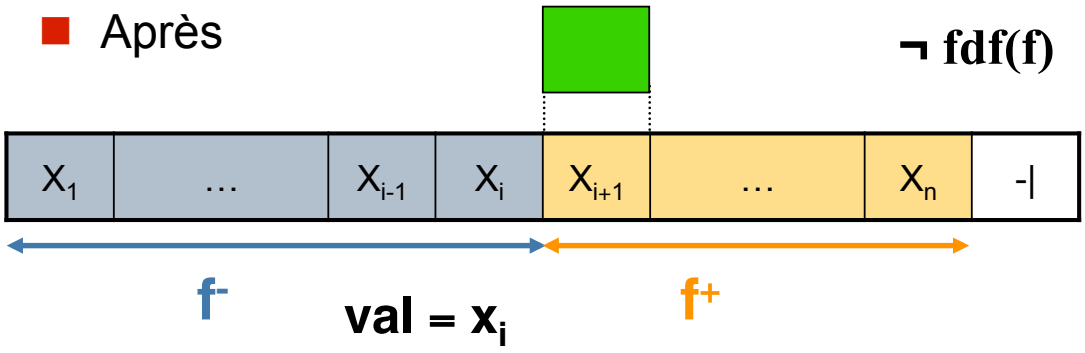
- Après l'appel de lire, **val = Df^-** où Df^- désigne le dernier élément du fichier f^-

Accès à l'élément courant

- Exemple 1
  - Avant

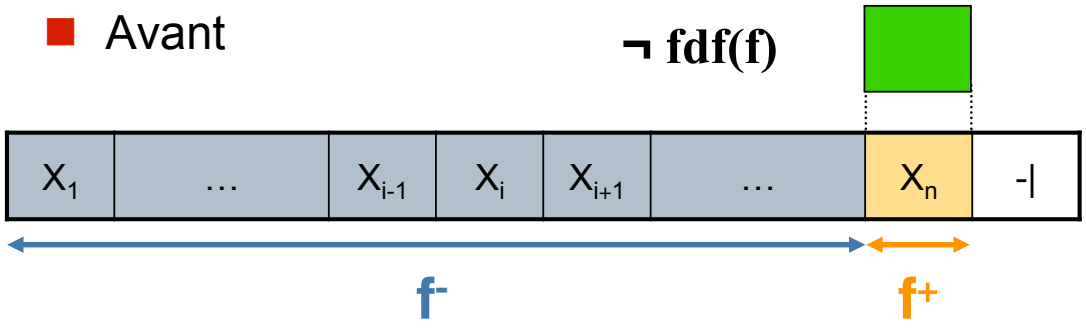


  - Après

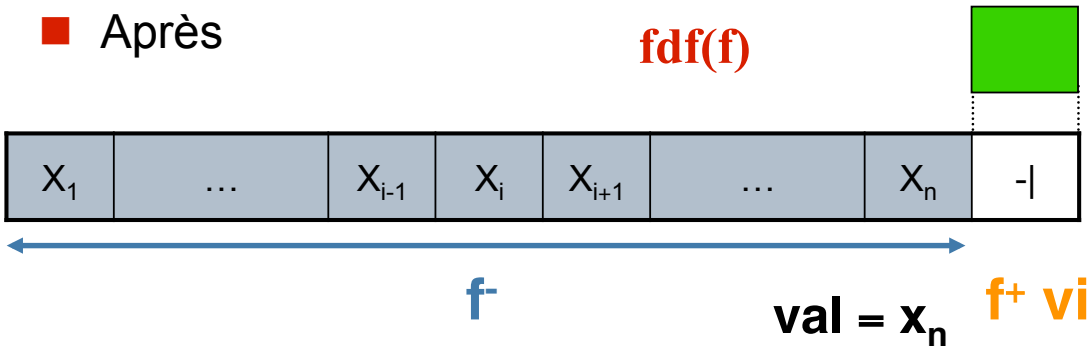


Accès à l'élément courant

- Exemple 2
  - Avant



  - Après



Notation f^{--}

- $f^{-} = f^{--} \,||\, val$
- f^{--} est le fichier f^{-} avant exécution de **lire(f, val)**
- Formellement :

$\{f = f_1^{i-1}\ ,\ f^{+} = f_i^{n}\ ,\ \neg\, fdf\ (f)\}$

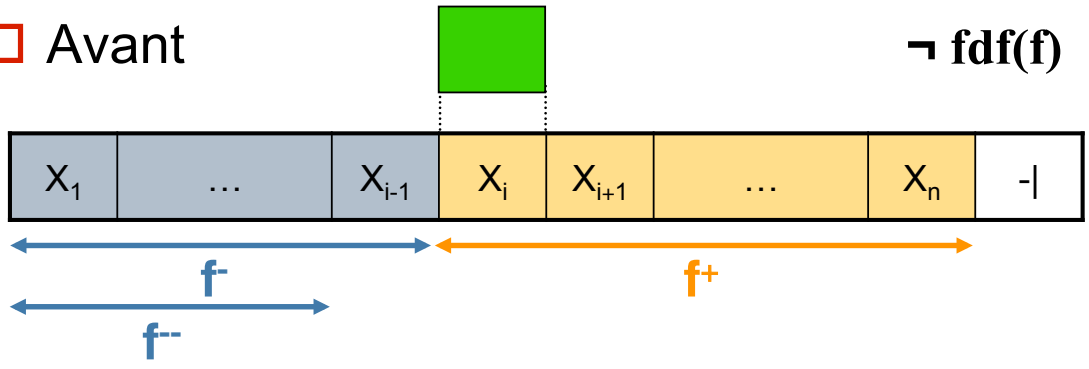
lire (f , val)

$\{val = x_i = Df\ ,f^{-} = f_1^{i-1},\ f = f_1^{i},\ (f^{+} = f_{i+1}^{n},\ \neg\, fdf\ (f),\ f^{+} \neq <>)$

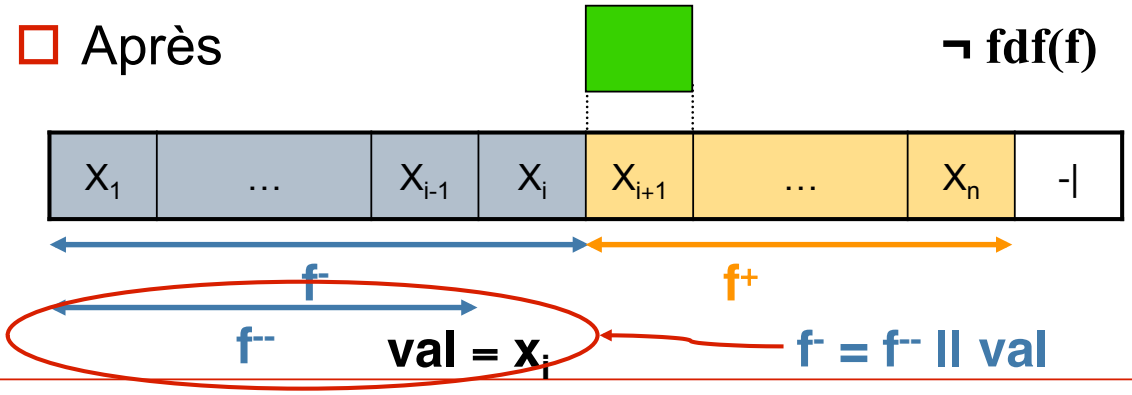
$v\ (fdf\ (f),\ f^{+} = <>)\}$

Notation f^{--} : illustration

- Avant



- Après



Exemple

- soit f = <24, -10, 53>

action	f^{--}	f^{-}	f^{+}	val	**fdf(f)**
				indef	indef
relire(f)		<>	<24, -10, 53>	indef	faux
lire(f, val)	<>	<24>	<-10, 53>	24	faux
lire(f, val)	<24>	<24, -10>	<53>	-10	faux
lire(f, val)	<24, -10>	<24, -10, 53>	<>	53	vrai
lire(f, val)	impossible fdf(f)	impossible fdf(f)	impossible fdf(f)		