



LIF5 - Algorithmique et programmation procédurale

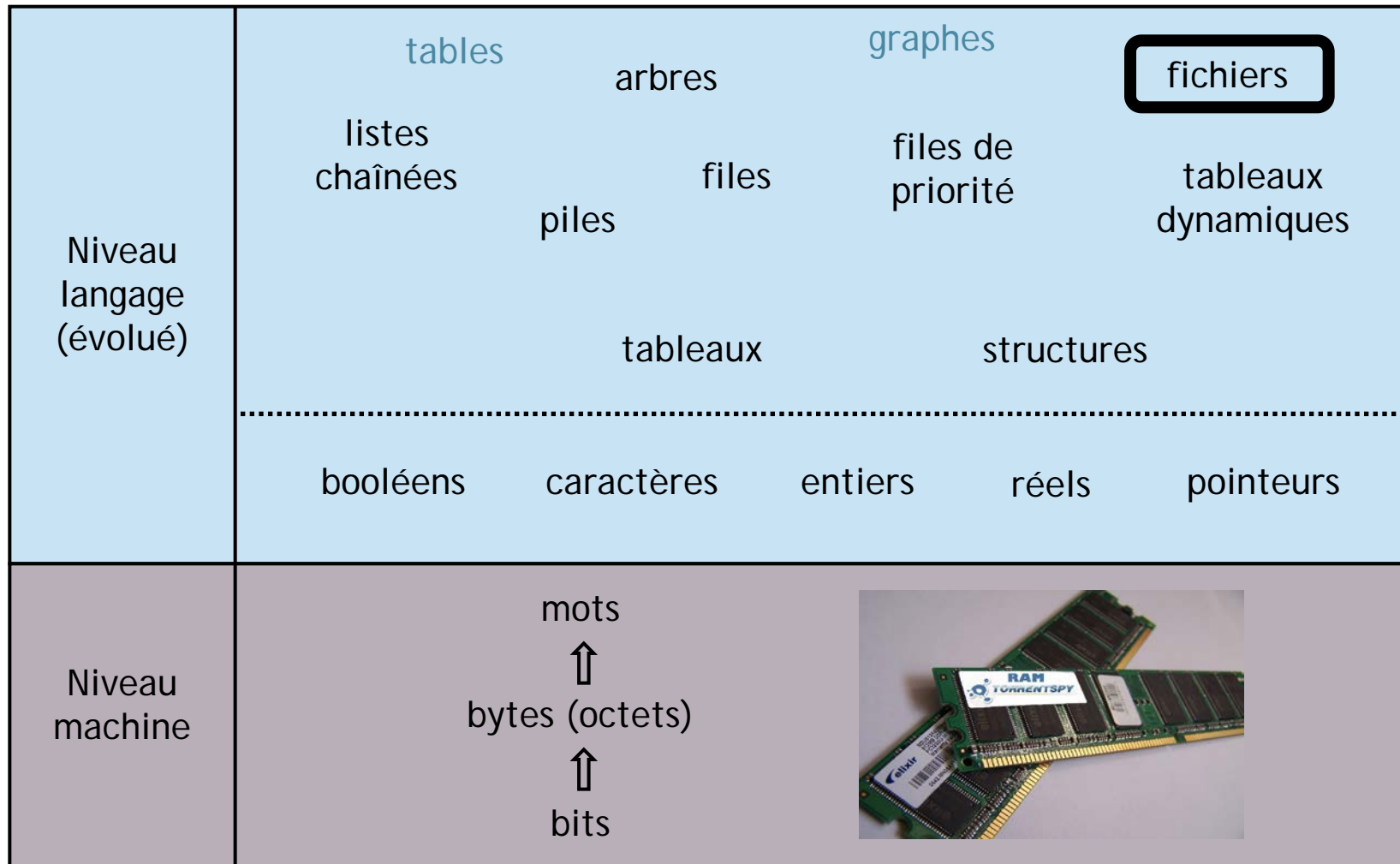
Carole Knibbe

Samir Akkouché

Chapitre 5

Les fichiers

Qu'est-ce que le type « fichier » ?



Sommaire

1. Généralités sur les fichiers
2. Manipuler des fichiers depuis un programme
 - notion de tampon (buffer)
 - mise en œuvre en C : le type FILE et les fonctions associées
3. Tri fusion sur fichier

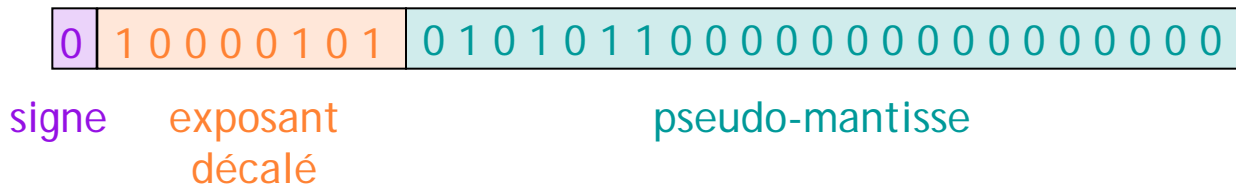
1. Généralités sur les fichiers

Différents types de fichiers : exemples

Format des informations	
« texte » (codes de caractères)	« binaire » (comme en mémoire vive)
carnet d'adresses personnelles fichiers .C fichiers de configuration Unix etc.	fichiers image fichiers mp3 fichiers doc fichiers pdf etc.

Fichiers « texte »

- Les octets du fichier ne contiennent que des codes de caractères
- Les données numériques doivent donc être converties en caractères
- Exemple : un nombre au format IEEE 754 simple précision



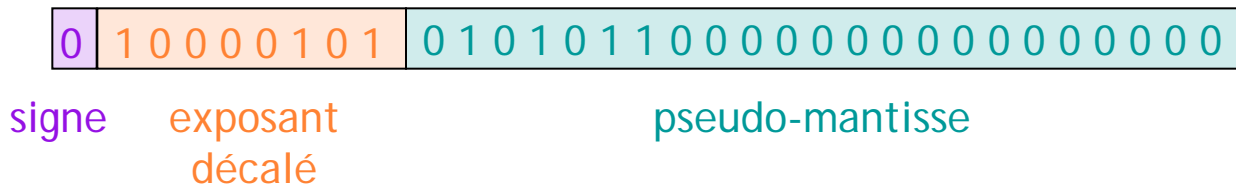
Décodage : nombre = $1,3359375 \cdot 2^6 = 85,5 = 8,55e1$

Ecriture dans le fichier (virgule fixe, 3 chiffres après la virgule) :

00111000	00110101	00101100	00110101	00110000	00110000
(56='8')	(53='5')	(44=',')	(53='5')	(48='0')	(48='0')

Fichiers « texte »

- Les octets du fichier ne contiennent que des codes de caractères
- Les données numériques doivent donc être converties en caractères
- Exemple : un nombre au format IEEE 754 simple précision



Décodage : nombre = $1,3359375 \cdot 2^6 = 85,5 = 8,55e1$

Ecriture dans le fichier (virgule flottante, 3 chiffres pour la mantisse) :

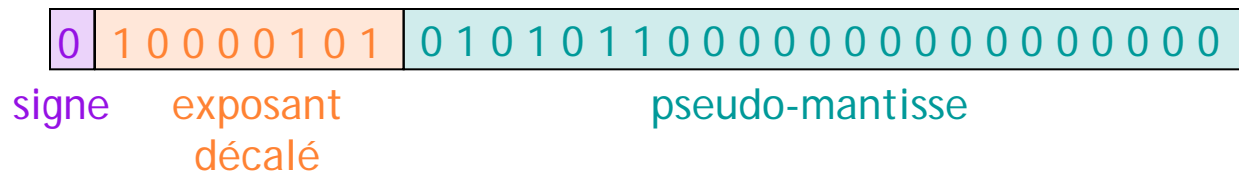
00111000	00101100	00110101	00110101	01100101	00110001
(56='8')	(44=',')	(53='5')	(53='5')	(101='e')	(49='1')

Fichiers « texte »

- Les octets du fichier ne contiennent que des codes de caractères
- Les données numériques doivent donc être converties en caractères
- Avantages :
 - portabilité
 - fichier lisible avec un simple éditeur de texte
- Inconvénients:
 - coût en temps induit par les conversions nécessaires
 - penser à écrire suffisamment de décimales pour ne pas perdre en précision numérique

Fichiers « binaires »

- On recopie l'information comme elle figure en mémoire vive
- Exemple :



Ecriture dans le fichier : 01000010 10101011 00000000 00000000

- Avantages :
 - fichier plus compact
 - lecture / écriture sans conversion = plus rapide
 - pas de formatage de présentation = pas de perte de précision
- Inconvénients :
 - fichier illisible par un simple éditeur de texte
 - il faut gérer les problèmes de portabilité

2. Manipuler des fichiers depuis un programme



Question

Doit-on se soucier du support physique des fichiers ?

Plusieurs niveaux d'abstraction

Bibliothèque stdio (C) :
fopen, fclose, fread, fwrite,
fprintf, fscanf, ...

Bibliothèque iostream (C++) :
open, close, <<, >>, ...

buffer

buffer

Appels système :
open, close, read, write (Unix)

Accès physique aux données
(déplacement tête de lecture, ...)

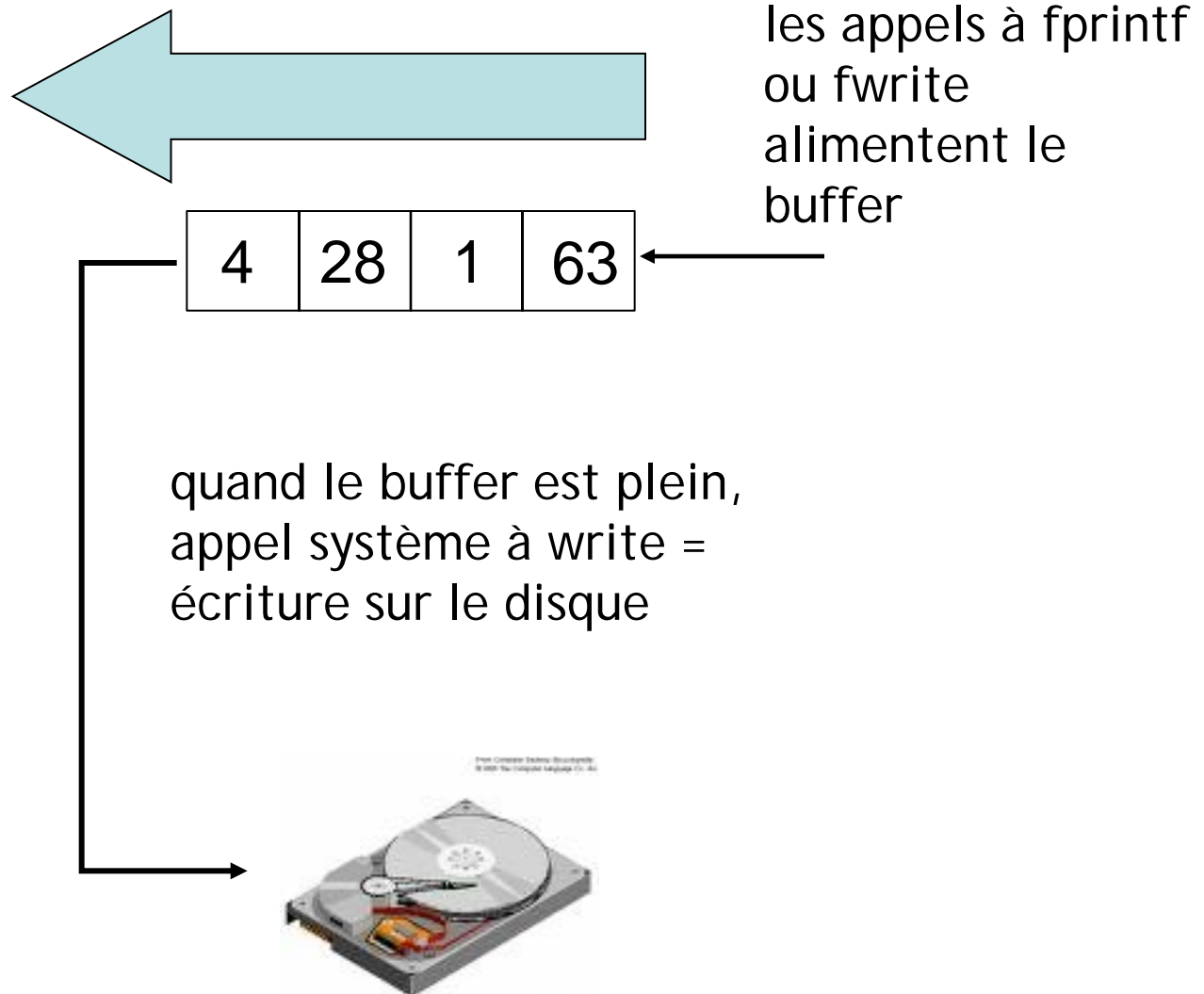




Notion de buffer

- Structures de type FIFO
 - = First in, First out
 - = file (ou 'queue' en anglais)
- Espace de stockage temporaire pour limiter le nombre d'appels système (lents !)

Buffer d'écriture



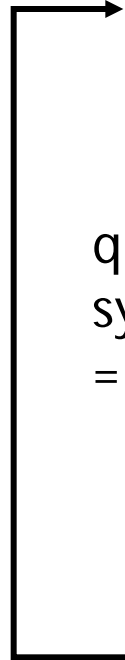
Buffer de lecture



les appels (réussis !)
à fscanf ou fread
volent le buffer

63	1	28	4
----	---	----	---

quand le buffer est vide, appel
système à read pour le remplir
= lecture depuis le disque





En C : le type FILE

```
/* Extrait de stdio.h */

struct __sbuf {
    unsigned char * __base;
    int __size;
};

typedef struct __sFILE {
    unsigned char * __p;
    short __file;
    struct __sbuf __bf;
    /* ... et d'autres membres */
} FILE;
```

- `_file` : numéro de fichier (file descriptor) alloué par l'OS lors de l'ouverture
- `_bf` : buffer = tableau d'octets
 - un int occupera 4 cases, un double 8, etc.
- `_p` : pointeur sur un élément du buffer

Ecrire un fichier en C : cas d'un fichier « binaire »

```
FILE * monFic;  
monFic = fopen("machin.jpeg", "wb");  
int e = 1356;  
fwrite(&e, sizeof(int), 1, monFic);  
...  
fclose(monFic);
```

- Demande à l'OS l'ouverture du fichier
- Association nom interne / nom externe
- Réservation de l'espace mémoire pour la struct FILE et pour son buffer

fopen(nom externe du fichier, mode d'ouverture)

Alloue un struct FILE dans le tas et retourne l'adresse de cet objet, ou NULL en cas d'échec de l'ouverture du fichier.



Ecrire un fichier en C : cas d'un fichier « binaire »

```
FILE * monFic;  
monFic = fopen("machin.jpeg", "wb");  
int e = 1356;  
fwrite(&e, sizeof(int), 1, monFic);  
...  
fclose(monFic);
```

Copier $1 * \text{sizeof}(\text{int})$ octets à partir de l'adresse de `e` et les recopier dans le buffer d'écriture.

Le contenu du buffer sera automatiquement écrit sur le disque (appel système) quand le buffer sera plein.

`fwrite(adresse 1er bloc à copier, taille bloc, nb blocs à copier, nom interne fichier)`

Retourne le nombre de blocs écrits avec succès

Ecrire un fichier en C : cas d'un fichier « binaire »

```
FILE * monFic;  
monFic = fopen("machin.jpeg", "wb");  
if (monFic == NULL) {  
    printf("Impossible d'ouvrir le fichier \n");  
    exit(EXIT_FAILURE);  
}  
  
int e = 1356; int valret;  
valret = fwrite(&e, sizeof(int), 1, monFic);  
if(valret < 1) {  
    printf("Erreur d'écriture \n");  
    exit(EXIT_FAILURE);  
}  
...  
fclose(monFic);
```

En réalité, il faut penser à la gestion des erreurs !

Erreurs d'écriture possibles : plus de place sur le volume, erreur matérielle, ...

Attention, la présence du buffer peut influencer sur le moment où ces erreurs sont détectées.

Ecrire un fichier en C : cas d'un fichier « texte »

```
FILE * monFic;  
monFic = fopen("toto.machin", "w");  
if (monFic == NULL) {  
    printf("Impossible d'ouvrir le fichier\n");  
    exit(EXIT_FAILURE);  
}  
int e = 1356; int valret;  
valret = fprintf(monFic, "%d", e);  
if(valret < 0) {  
    printf("Erreur d'écriture \n");  
    exit(EXIT_FAILURE);  
}  
...  
fclose(monFic);
```

Mettre les codes ASCII de '1', '3', '5', et '6' dans le buffer.

Le contenu du buffer sera réellement écrit sur le disque (appel système) quand le buffer sera plein.

`fprintf(nom interne fichier, code(s) de format, valeur1, valeur2, ...)`
Retourne le nombre de caractères écrits, ou une valeur négative en cas d'erreur

Lire un fichier en C : cas d'un fichier « binaire »

```
FILE * monFic;
int e, valret;
monFic = fopen("data.bin", "rb");
if(monFic == NULL) {
    printf("Impossible d'ouvrir le fichier\n ");
    exit(EXIT_FAILURE);
}

valret = fread(&e, sizeof(int), 1, monFic);
if(valret != 1) {
    printf("Fin de fichier ou erreur de lecture \n");
}
/* ... */
fclose(monFic);
```

- demander à l'OS d'ouvrir data.txt en lecture,
- associer le nom interne au nom externe
- réserver l'espace mémoire pour le buffer

Lire un fichier en C : cas d'un fichier « binaire »

```
FILE * monFic;
int e, valret;
monFic = fopen("toto.pif ", "rb");
if(monFic == NULL) {
    printf("Impossible d'ouvrir le fichier\n");
    exit(EXIT_FAILURE);
}

valret = fread(&e, sizeof(int), 1, monFic);
if(valret != 1) {
    printf("Fin de fichier ou erreur de lecture \n");
}
/* ... */
fclose(monFic);
```

- lire les 4 premiers octets du tampon (il est automatiquement rempli d'après le fichier s'il est vide),
- les copier dans la variable e,
- puis les enlever du tampon

fread(adresse 1er bloc à remplir, taille bloc, nb blocs à remplir, nom interne fichier)

Retourne le nombre de blocs lus avec succès

Lire un fichier en C : cas d'un fichier « texte »

```
FILE * monFic;
int e, valret;
monFic = fopen("data.txt", "r ");
if(monFic == NULL) {
    printf("Impossible d'ouvrir le fichier\n");
    exit(EXIT_FAILURE);
}

valret = fscanf(monFic, " %d ", &e);
if(valret != 1) {
    printf("Fin de fichier ou erreur de lecture \n");
}
/* ... */
fclose(monFic);
```

- lire les premiers caractères du tampon jusqu'à tomber sur un espace ou une fin de ligne,
- convertir ces caractères vers l'entier correspondant
- mettre ce nombre dans la variable e,
- puis enlever les caractères traités du tampon

fscanf(nom interne fichier, code(s) de format, adresse variable 1, adresse variable 2, ...)

Retourne le nombre de variables affectées avec succès



Lire un fichier en C : le piège de feof

- La structure FILE contient un indicateur de fin de fichier qui peut être examiné avec la fonction « feof »
- Il est alors tentant d'écrire :

```
int nb_passages = 0;
while( !feof(monFic) ) {
    fscanf(monFic, "%d", &monInt);
    printf("Je viens de lire %d\n", monInt);
    nb_passages ++;
}
```

- Mais ce code contient en fait un bug : on boucle une fois de trop, parce que feof n'anticipe pas la fin de fichier
- La lecture des derniers octets du fichier ne permet pas d'en détecter la fin, il faut essayer de lire au moins un octet inexistant

Lire un fichier en C : le piège de feof

- Solution :
 - ne pas utiliser feof comme condition de bouclage
 - utiliser la valeur de retour de fscanf ou fread
 - on n'exécute le reste de la boucle que si la lecture a réussi

```
int nb_passages = 0;
while(fscanf(monFic, "%d", &monInt) == 1) {
    printf("Je viens de lire %d\n", monInt);
    nb_passages ++;
}

if(feof(monFic)) printf("Fin de fichier \n");
else printf("Autre probleme \n");
```

3. Tri fusion sur fichier


Cas de données encombrantes

- Parfois, le nombre d'objets à trier est tel qu'on ne peut pas les stocker tous en mémoire vive
- Exemples :
 - articles d'un gros catalogue de vente par correspondance
 - données d'un recensement national
- Il faut des algorithmes de « tri externe », c'est-à-dire capables de travailler en ne stockant en mémoire vive que des petites parties des fichiers

Tri externe : caractéristiques

- Pendant le tri, seule une partie des données se trouve en mémoire vive, le reste étant stocké sur disque ou sur bande magnétique
- Accéder aux données est donc beaucoup plus coûteux que de les comparer ou de faire des opérations arithmétiques dessus, par exemple
- Il peut y avoir des restrictions fortes sur l'accès
 - ex. si bandes, accès uniquement séquentiel

Tri externe : critères pour les algorithmes

- Critère 1 : minimiser le nombre de fois où l'on accède à un élément
- Critère 2 : accéder séquentiellement aux éléments
- Les deux algorithmes de tri externe les plus courants sont :
 - le tri fusion (merge sort) 
 - le tri par paquets (bucket sort)

Notion de monotonie

- Une monotonie est une sous-séquence d'éléments triés
- Toute sous-séquence de taille 1 est trivialement une monotonie
- Exemple :

56 12 16 23 28 6 2 1 57 59 45



11 monotonies de longueur 1

Notion de monotonie

- Une monotonie est une sous-séquence d'éléments triés
- Toute sous-séquence de taille 1 est trivialement une monotonie
- Exemple :

56 12 16 23 28 6 2 1 57 59 45



The diagram shows a sequence of numbers: 56, 12, 16, 23, 28, 6, 2, 1, 57, 59, 45. Below the numbers, there are five orange curly brackets, each spanning two numbers. The first bracket is under 12 and 16. The second is under 16 and 23. The third is under 23 and 28. The fourth is under 1 and 57. The fifth is under 57 and 59. These brackets represent increasing subsequences of length 2.

5 monotonies de longueur 2

Notion de monotonie

- Une monotonie est une sous-séquence d'éléments triés
- Toute sous-séquence de taille 1 est trivialement une monotonie
- Exemple :

56 12 16 23 28 6 2 1 57 59 45



3 monotonies de longueur 3

Notion de monotonie

- Une monotonie est une sous-séquence d'éléments triés
- Toute sous-séquence de taille 1 est trivialement une monotonie
- Exemple :

56 12 16 23 28 6 2 1 57 59 45



Monotonies maximales :

1 de longueur 4


1 de longueur 3



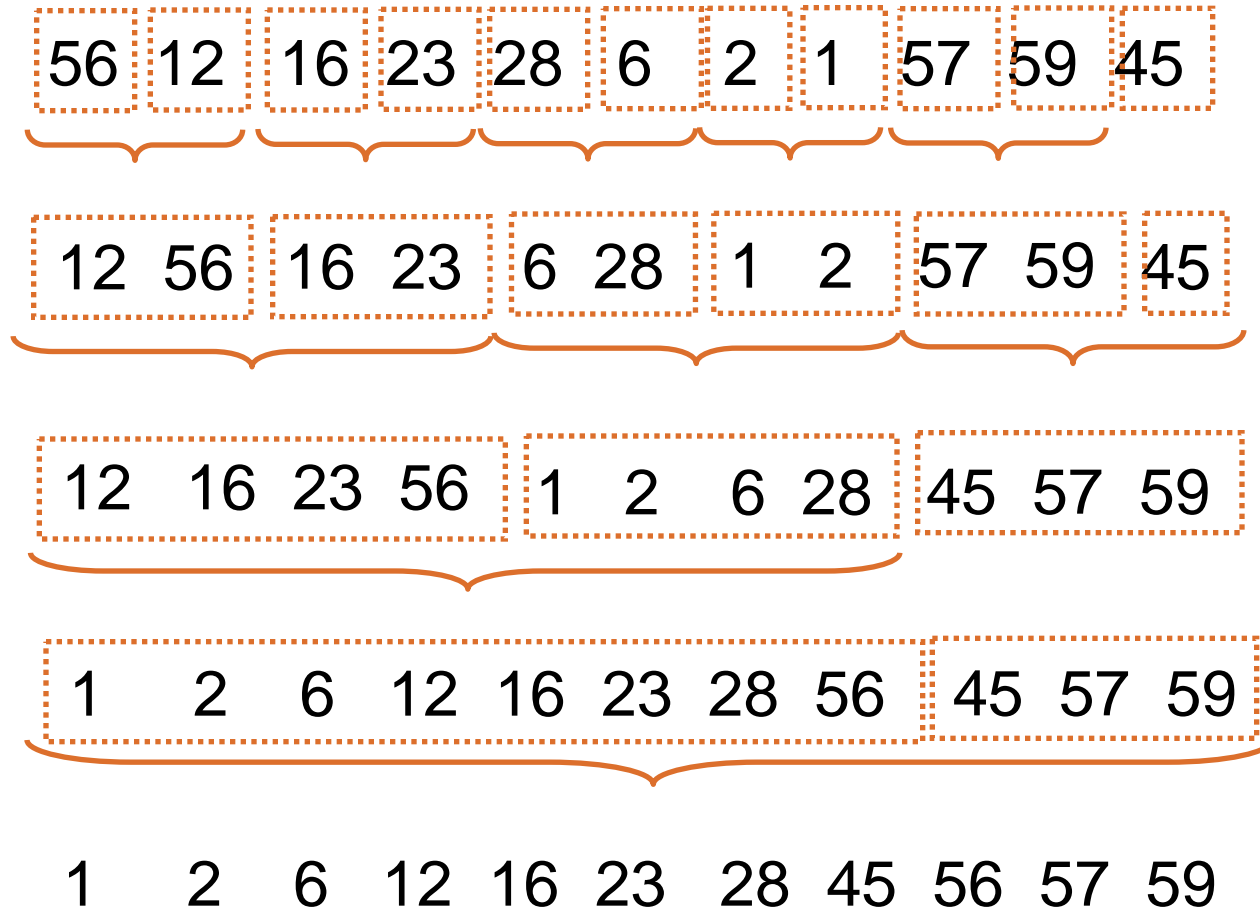
Fusion de deux monotonies

- Fusionner deux monotonies = construire une nouvelle monotonie qui comprenne tous les éléments des deux monotonies initiales
 - on prend le plus petit des 2 éléments accessibles (= ceux en tête de chaque monotonie) et on l'écrit dans la nouvelle monotonie
 - on passe à l'élément suivant sur la monotonie dont on a enlevé l'élément de tête
 - on recommence ce processus jusqu'à épuisement d'une monotonie
 - on prend tout le reste de l'autre monotonie et on l'écrit dans la nouvelle monotonie
- A chaque instant, on n'a besoin en mémoire que des deux éléments à comparer
- On accède bien séquentiellement aux éléments d'une monotonie donnée

Principe du tri fusion

- Approche top-down (utilisée en tri interne)
- Approche bottom-up (utilisée en tri externe) 
 1. Fusionner les paires d'éléments successifs pour obtenir une séquence de $n/2$ monotonies de longueur 2
 2. Fusionner les paires de monotonies successives pour obtenir une séquence de $n/4$ monotonies de longueur 4
 3. Continuer jusqu'à obtenir une seule monotonie de longueur N

Principe du tri fusion : exemple





Principe du tri fusion : question

- Approche bottom-up (utilisée en tri externe)
 1. Fusionner les paires d'éléments successifs pour obtenir une séquence de $n/2$ monotonies de longueur 2
 2. Fusionner les paires de monotonies successives pour obtenir une séquence de $n/4$ monotonies de longueur 4
 3. Continuer jusqu'à obtenir une seule monotonie de longueur N

Il n'est pas conseillé de réaliser le tri fusion sur fichier avec seulement 2 fichiers. Pourquoi, à votre avis ?

Tri fusion sur fichier

- Approche bottom-up, avec 3 fichiers qui doivent être sur 3 supports extérieurs distincts
- Phase d'éclatement : on répartit les monotonies contenues dans le fichier X dans deux fichiers
 - 1 sur 2 va dans le fichier A
 - 1 sur 2 va dans le fichier B
- Phase de fusion :
 - on fusionne la 1e monotonie du fichier A avec la 1e monotonie du fichier B, en écrivant la monotonie résultante dans le fichier X (on écrase l'ancien contenu du fichier X)
 - idem avec les 2e monotonies des fichiers A et B
 - et ainsi de suite jusqu'à la fin des fichiers A et B
- On recommence la phase d'éclatement du fichier X, etc.

Points importants du chapitre

- Fichier texte / fichier binaire : avantages et inconvénients
- Abstraction du support physique grâce à l'OS
- Flux bufferisés pour limiter les appels systèmes
- Gestion des erreurs et des fins de fichiers
- Tri de grands fichiers : il faut des algorithmes spécifiques dits de « tri externe »