CI 2 : Algorithmique & Programmation

Chapitre 5 – Lecture et écriture des fichiers

Savoir

Savoirs :
– **

1 Pourquoi utiliser des fichiers ?

L'ordinateur sert à traiter de l'information qui peut entrer et sortir de différentes façons :
– le clavier, la souris et l'écran, pour l'interface utilisateur du programme ou le shell ;
– les fichiers, pour les supports mémoire ;
– les interfaces de communication (réseau, port USB, port série, etc).

Le clavier et la souris ne permettent pas d'entrer une grosse masse d'information. De même l'écran montre une petite partie de l'information. En revanche les fichiers permettent de décupler les possibilités de traitement d'information des programmes. Un roman de 200 pages contient environ 500 000 caractères, et donc tient en un fichier de 500 ko environ.

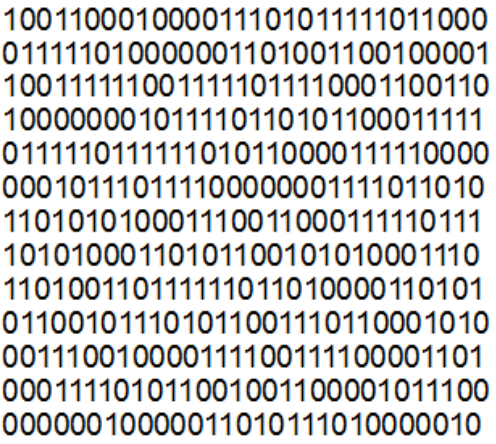
Le réseau permet d'échanger des informations sous deux formes :
– sous forme de fichiers échangés, qui sont ensuite stockés sur le disque dur ;
– sous forme de flux de données. Ces données sont traitées par les cartes interfaces et mises à disposition des programmes par l'OS sous une forme proche des fichiers.

Il est donc intéressant de savoir lire et écrire sur les fichiers.

1.1 Deux familles de fichiers

Les fichiers sont tous écrits en binaire. Il est néanmoins possible de les séparer en deux familles :

- les fichiers binaires qui nécessitent de connaître le format binaire d'écriture des données pour être lus ;
- les fichiers texte qui contiennent des caractères uniquement, et qui peuvent s'ouvrir sur un éditeur de texte.



Exemple

Les fichiers binaires :
- images et documents (bmp, png, jpg, pdf, doc, etc) ;
- son et vidéo (wav, mp3, mp4, etc) ;
- exécutables (.exe) ;
- archives compressées (zip, 7z, gz).

Les fichiers texte :
- pages Web (html, css, etc) ;
- fichier journal (log), Script shell (bat) ;
- images vectorielles (svg) ;
- programmes Python ou Scilab (py, sce) ;
- les fichiers de données texte (txt, data, etc) ;
- les fichiers texte formatés (xml).

Les fichiers texte compressés :
- les fichiers bureautique (odt, ods, docx, xlsx).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tra
xmlns="http://www.w3.org/1999/xhtml" xml:lang="f
name="description" content="Union des Professeur
<meta
http-equiv="Content-Type" content="text/html; ch
name="generator" content="SPIP 3.0.11-dev [20322
rel="shortcut icon" href="squelettes/favicon.ico
rel="alternate" type="application/rss+xml" title
var box_settings={tt_img:true,sel_g:"#documents_
</script> <link
rel='stylesheet' href='local/cache-css/3d23ccf53
rel="alternate" type="application/json+oembed" h
rel="alternate" type="text/xml+oembed" href="htt
</head><body
class="page_accueil large"><div
id="page">
<div
id="entete" class="zone">
```

1.2 Avantages et inconvénients

	Les fichiers binaires	Les fichiers texte
Avantages	Moins volumineux Indépendants des standards d'encodage des caractères dans les OS	Interprétables par l'homme Permettent des d'échanges plus simples entre logiciels Ne nécessitent généralement pas de bibliothèques
Inconvénients	Moins faciles à lire Nécessitent des bibliothèques pour les ouvrir	Plus volumineux Dépendants du format d'encodage des caractères

2 Fichiers binaires

2.1 Analyse d'un fichier binaire : BMP

On peut ouvrir un fichier binaire avec un éditeur héxadécimal.

Les deux premiers caractères de cette exemple "42" représentent en hexadécimal le codage sur les 8 bits correspondants (celui-ci apparait en bas de la fenêtre quand on se place sur le "42").

Un fichier BMP est un format très simple pour mémoriser les images :

- signature (BM, BA, CI, ...) ;
- taille du fichier (4o) ;
- champ réservé (4o) ;
- offset de début données (4o) ;
- taille de l'entête (4o) ;
- largeur de l'image (4o) ;
- hauteur de l'image (4o) ;
- nombre de plans (2o) ;
- profondeur : 1 à 32 (2o) ;
- type compression (4o) ;
- *etc.*

Les couleurs commencent à l'octet 122=0x7A (octet en jaune) :

- blanc (ff ff ff) ;
- blanc (ff ff ff) ;
- blanc (ff ff ff) ;
- blanc (ff ff ff) ;
- gris clair (e9 ec ef) ;
- gris (ce cc c4) ;
- gris foncé (a0 97 7c)...

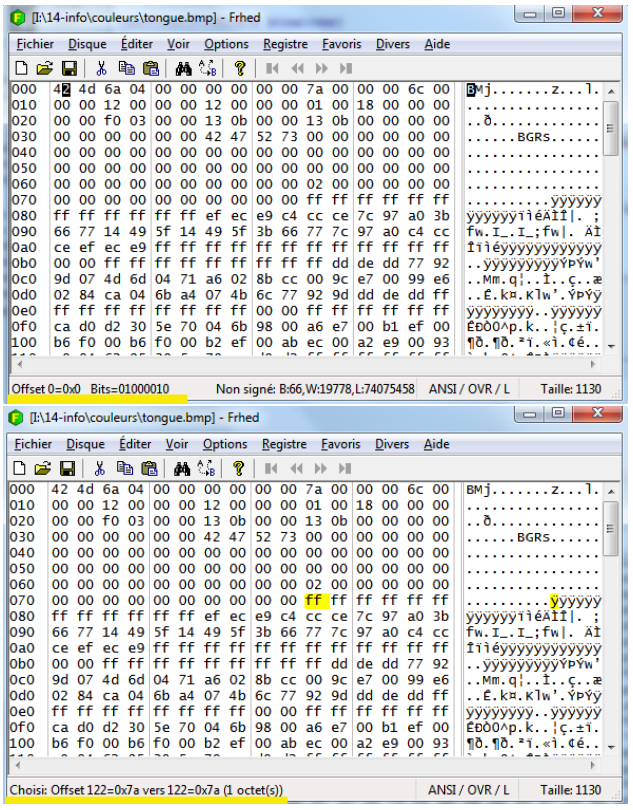


FIGURE 1 – Ouverture d'un fichier binaire avec un éditeur hexadécimal

2.2 Ouvrir des fichiers binaires

Les formats étant généralement assez complexes et variés, les fichiers binaires sont ouverts via des **librairies**. Ces librairies proposent des commandes toutes prêtes. Par exemple pour les images :

- Python : librairie PIL (Python Imaging Library) ;
- Scilab inclut des commandes pour les images

On code très rarement les commandes permettant d'ouvrir les fichiers binaires. Pour lire tout de même un fichier binaire on utilise la fonction open, disponible sans avoir besoin de rien importer. Elle prend en paramètre :

- le chemin (absolu ou relatif) menant au fichier à ouvrir ;
- le mode d'ouverture.

Le mode est donné sous la forme d'une chaîne de caractères. Voici les principaux modes :

- 'r' : ouverture en lecture (Read) ;
- 'w' : ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé,

– 'a' : ouverture en écriture en mode ajout (Append). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est créé.

On peut ajouter à tous ces modes le signe b pour ouvrir le fichier en mode binaire.

```python
# Lecture d'un fichier binaire
f = open("tongue.bmp", "rb")

while True:
        bytes = f.read(1) # lecture d'un octet
        if bytes == "":
            break;
        # Affichage de l'octet lu en hexadécimal :
        print "%02X " % ord(bytes[0]),

f.close()
```

3 étapes :

– Ouverture du fichier "tongue.bmp", en lecture mode binaire ("rb").
– Boucle sur chaque octet pour lire et afficher. La méthode read renvoie le contenu du fichier, que l'on capture dans bytes.
– Fermeture du fichier : n'oubliez pas de fermer un fichier après l'avoir ouvert. Si d'autres applications, ou d'autres morceaux de votre propre code, souhaitent accéder à ce fichier, ils ne pourront pas car le fichier sera déjà ouvert. C'est surtout vrai en écriture, mais prenez de bonnes habitudes. La méthode à utiliser est close.

2.3 Écrire dans des fichiers en binaire

```python
# Écriture d'un fichier binaire
f=open("TP/monFichier.bin","wb")
f.write("Du texte")
f.write(int8(83))
f.write(int8(76))
f.write(float32(2.3))
f.close()
```

3 étapes :

– Il faut ouvrir le fichier avant tout. Ouverture du fichier "monFichier.bin", en écriture mode binaire ("wb").
– Écriture d'octets (caractères, nombres entiers ou flottants) : On utilise la méthode write. Deux modes sont possibles : le mode w ou le mode a. Le premier écrase le contenu éventuel du fichier, alors que le second ajoute ce que l'on écrit à la fin du fichier. Ces deux modes créent le fichier s'il n'existe pas.
– Fermeture du fichier.

3 Fichiers texte

Un fichier texte brut ou fichier texte simple est un fichier dont le contenu représente uniquement une suite de caractères. Bien qu'on l'oppose ici aux fichiers binaires il est lui aussi codé en binaire sur l'ordinateur. Cependant ce codage est basé sur une norme connue de tous les éditeurs de texte afin de traduire le fichier en une suite de caractères "imprimables". Les caractères considérés sont généralement les caractères imprimables, d'espaces et de retours à la ligne. La notion de fichier texte est subjective et dépend notamment des systèmes de codage de caractère considérés. Ainsi si l'encodage est inconnu, un texte brut quelconque est inexploitable.

Il existe de nombreux standards de codage, dont l'American Standard Code for Information Interchange ASCII. Cette norme ancienne créée pour gérer des caractères latins non accentués (nécessaire pour écrire en anglais) est à la base de nombreux codages de caractères.

L'ASCII permet de coder 128 caractères numérotés de 0 à 127 et peut donc être codé sur 7 bits. Cependant, les ordinateurs travaillent la plupart du temps en multiple de 8 bits, le huitième bit est mis à 0. On a donc un octet par caractère.

L'absence d'accents rend cette norme insuffisante à elle seule, ce qui rend nécessaire l'utilisation d'autres encodages : UTF-8 par exemple (UCS transformation format 8 bits) dans lequel chaque caractère est représenté par un index et son codage binaire donné par une table. Les 128 premiers caractères ont un codage identique en ASCII et UTF8 (par exemple "A" a pour code ASCII 65 et se code en UTF-8 par l'octet 65) puis d'autres caractères sont ajoutés.

Code Hexadécimal poids fort →

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

↑ Code Hexadécimal poids faible

FIGURE 2 – Table des caractères ASCII

3.1 Lecture d'un fichier sous Python

Les fichiers texte sont écrits (en binaire) de façon à respecter un des codes standards de caractères (utf8, iso-8859, ASCII...). Ils peuvent s'ouvrir sur un éditeur de texte, ce qui permet de lire ou modifier le contenu beaucoup plus facilement qu'en binaire.

EXEMPLE : fichier de mesure sur l'axe Emericc : *Mesure_axe_Emericc.txt*

OBJECTIF : Lire les données (paramètres et mesures) et tracer les courbes.

- 12 lignes de paramètres ;
- 100 lignes de données ;
- 9 lignes de paramètres.

Lecture des noms :

```python
# Lecture d'un fichier texte ligne à ligne
# Ouverture fichier
f=open("TP_Fichiers/Mesure_axe_Emericc.txt","r")
ligne = f.readline()                    # lecture d'une ligne
# Affichage pour vérification
print "%s" % ligne
ligne=ligne.rstrip("\n\r")              # suppression retour chariot
noms_grandeurs=ligne.split("\t")        # découpage aux tabulations
noms_grandeurs=noms_grandeurs[1:3] # suppression de "noms"
```

```
Nom Position    Consigne variateur
Unite Ox    s       s
Unite Oy    mm
Delta (Ox) 0,01  0,01
Delta (Oy) -1,00-1,00
Minimum (Ox)    0,00 0,00
Maximum (Ox)    0,80 0,80
Minimum (Oy)    0,00 -66,00
Maximum (Oy)    7,28 127,00
Indice du minimum (Oy)    0,00  17,00
Indice du maximum (Oy)    17,00       1,00
Nombre de points    100  100
0       0,00 8,00
1       0,06 127,00
2       0,23 127,00
3       0,51 127,00
4       0,88 119,00
5       1,34 107,00
6       1,91 92,00
7       2,52 76,00
[...]
97      4,94 1,00
98      4,94 1,00
99      4,94 1,00
Position
Amplitude de l'échelon      5,00
Gain proportionnel    20,00
Manipulation      Correcteur proportionnel

Consigne variateur
Amplitude de l'échelon      5,00
Gain proportionnel    20,00
Manipulation      Correcteur proportionnel
```

Lecture du nombre de points :

```python
for i in range(10):
    ligne = f.readline()    # saut de 10 lignes

ligne = f.readline()                    # lecture d'une ligne
print "%s" % ligne                      # affichage pour vérification
ligne=ligne.rstrip("\n\r")              # suppression retour chariot
ligne_nbpoints=ligne.split("\t")        # découpage aux tabulations
nb_points=int(ligne_nbpoints[1])        # conversion en entier
```

Lecture des données :

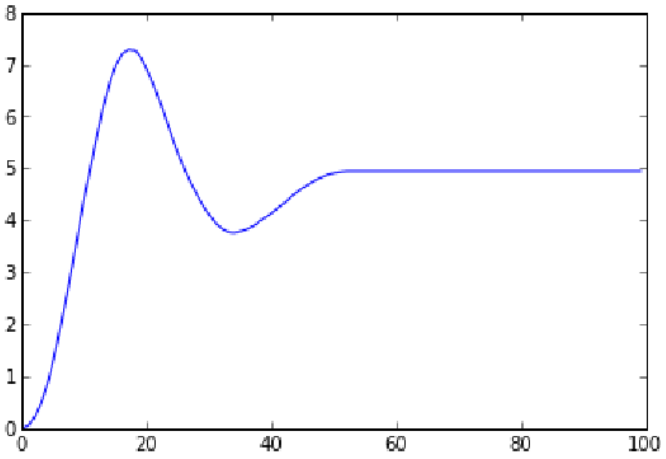
```python
numero=[] ; position=[] ; consigne=[] # initialisation tableaux

for i in range(nb_points):
    ligne = f.readline()                # lecture d'une ligne
    ligne=ligne.rstrip("\n\r")          # suppression retour chariot
    ligne=ligne.replace(",",".")        # changement , en .
    ligne_data=ligne.split("\t")        # découpage aux tabulations
    numero.append(int(ligne_data[0]))
    position.append(float(ligne_data[1]))     # Ajout aux tableaux
    consigne.append(float(ligne_data[2]))
```

Fermeture du fichier et Tracé de la courbe

```python
f.close()          # Fermeture fichier

plot(position)     # Tracé de la courbe de position
```



3.2 Lecture d'un fichier sous Scilab

De la même façon que Python, Scilab permet de lire des fichiers. La syntaxe est proche :

```scilab
// Ouverture du fichier et lecture ligne a ligne
fic=mopen("Mesure_axe_Emericc.txt","r");
ligne=mgetl(fic,1)
// Decoupage a la tabulation = caractere ascii 9
noms_grandeurs=strsplit(ligne, ascii(9))
noms_grandeurs=noms_grandeurs(2:3)
for i=1:10
    ligne =mgetl(fic,1);
end

ligne =mgetl(fic,1)
ligne_nbpoints= strsplit(ligne, ascii(9))
nb_points=int(ligne_nbpoints(2))
nb_points=msscanf(ligne_nbpoints(2),"%i")
```

Lecture des données et affichage de la courbe.

```
numero=[];position=[];consigne=[];
for i=1:nb_points
    ligne =mgetl(fic,1);
    ligne = strsubst(ligne,",",".");
    [n,numero(i), position(i),consigne(i)]=msscanf(ligne,"%d\t%f\t%f");
end
mclose(fic);                    // Fermeture du fichier
plot(position)
```

3.3 Cas des données formatées

Le tableau de données est « formaté », c'est-à-dire qu'il présente une structure identique à chaque ligne.

Python possède des outils de lecture automatique de ce type de tableau : numpy.loadtxt()

```
a=loadtxt("Fichier.txt",
                dtype={
                        'names': ('numero', 'position', 'consigne'),
                        'formats': ('i2', 'f4', 'f4')},
                delimiter='\t')
```

Int	float	float
0	0,00	8,00
1	0,06	127,00
2	0,23	127,00
3	0,51	127,00
4	0,88	119,00
5	1,34	107,00
6	1,91	92,00
7	2,52	76,00
...	...	...
97	4,94	1,00
98	4,94	1,00
99	4,94	1,00

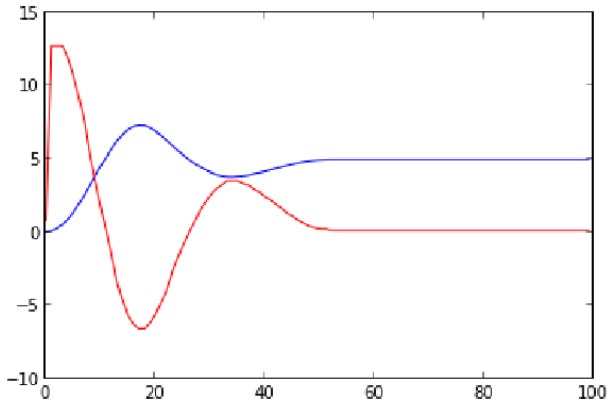
Pour récupérer les données :

```
a['numero']                     #liste de valeurs de la colonne N
a['numero'][10]                 #11ème élément
```

```
a,b=loadtxt("Fichier.txt",
                usecols = (0,2),
                dtype={
                        'names': ('numero', 'consigne'),
                        'formats': ('i2', 'f4')},
                delimiter='\t',
                unpack=True)
```

- usecols : colonnes à utiliser dans le fichier
- dtype : type de données à lire
- Names : nom
- format : entier sur 2o, flottant sur 4o, strings...
- délimiter : séparateur des données
- unpack : permet de séparer les colonnes → a,b=...

```python
plot(a['numero'],a['position'],'b',
                a['numero'],a['consigne']/10,'r')
```



3.4 Lecture d'un fichier texte formaté sous Scilab

De la même façon que Python, Scilab permet de lire des fichiers formatés.

La synthaxe est proche (en plus simple quand même...).

```
// Lecture de donnees formatees
fic=mopen("Mesure_axe_Emericc_formate.txt","r");
T=mfscanf(-1,fic,'%d\t%f\t%f')
plot(T(:,1),[T(:,2),T(:,3)/10])
mclose(fic);                    // Fermeture du fichier
```

3.5 Écriture d'un fichier texte sous python

L'écriture d'un fichier texte est très simple sous python :

```python
# Écriture d'un fichier texte ligne à ligne
f=open("TP_Fichiers/monFichier.txt","w") # Ouverture du fichier
f.write("La température est froide l'hiver.\n")
f.write("Il fait {:f} degrés.".format(10))
f.close()          # Fermeture du fichier
```

Et pour un fichier formaté :

```python
# Écriture d'un fichier formaté
f=open("TP/monFichier.txt","w") # ouverture fichier
x=linspace(-20,20,100)
y=sin(x)/x
for i in range(0,len(x)):
    f.write("{:d} \t {:f} \t {:f}\n".format(i,x[i],y[i]))
f.close()          # Fermeture du fichier
```

3.6 Écriture d'un fichier texte sous Scilab

L'écriture d'un fichier texte, formaté ou pas, est très simple :

```
// Ecriture de donnees formatees ou non ...
fic=mopen("monFichier.txt","w");
mfprintf(fic,"Voici mon fichier de point\n")
mfprintf(fic,"Nombre de points : %d\n",100)
x=-20:40/99:20;
y=sin(x)./x;
mfprintf(fic,'%d\t%f\t%f\n',[1:100]',x',y')
mclose(fic); // Fermeture du fichier
```

4 Enregistrer un objet dans un fichier : Module Pickle

Dans Python comme dans beaucoup de langages de haut niveau, on peut enregistrer les objets dans un fichier. Lorsque l'objectif est de sauver des objets python pour les récupérer plus tard sous python, il est pratique d'utiliser Pickle.

Alors que les fonctions utilisées dans ce cours ne nécessitaient pas l'importation de bibliothèque, il faut penser ici à importer Pickle.

Soit v une variable quelconque,

Sauvegarde :

```
import pickle
fic=open("nao.pick","wb"")
pickle.dump(v,fic)
fic.close()
```

On utilise la méthode dump pour enregistrer l'objet.

Une fois ce code exécuté un fichier nao.pick aura été créé avec les données correspondantes à l'intérieur.

Pour stocker plusieurs variables, il suffit d'appeler plusieurs fois la fonction pickle.dump() pour chaque variable.

Lecture :

```
import pickle
fic=open("nao.pick","rb"")
v=pickle.load(fic)
fic.close()
```

Pour recharger ces variables, il faut appeler autant de fois la fonction pickle.load(). Les variables sont restituées dans le même ordre.

Références

[1] Marc Derumeaux et Damien Iceta, *Les fichiers. Apprendre à lire et à écrire*, UPSTI.